Universidad Autónoma de Madrid
Escuela politécnica superior

Grado en Ingeniería Informática

Trabajo Fin de Grado

ANÁLISIS DE DATOS GEO-ESPACIALES UTILIZANDO LA PLATAFORMA BLUEMIX

Autor: Alexandre Cándido Henrique Porcides
Tutor: Francisco Javier Gómez Arribas

JUNIO 2016

ANÁLISIS DE DATOS GEO-ESPACIALES UTILIZANDO LA PLATAFORMA BLUEMIX

Autor: Alexandre Cándido Henrique Porcides
Tutor: Francisco Javier Gómez Arribas

Cátedra UAM IBM
Dpto. de TEC
Escuela Politécnica Superior
Universidad Autónoma de Madrid
JUNIO 2016

Resumen

Actualmente estamos sufriendo una transformación en la que los dispositivos se comunican con dispositivos, y dichas comunicaciones están creando nuevos modelos de negocio, productos y compañías. Hace aproximadamente 20 años, internet se utilizaba únicamente como una herramienta para buscar información. En los últimos 10 años ese paradigma se ha ido transformando al ámbito social, transaccional y móvil. Se estima que el número de dispositivos conectados a internet en el año 2020 será de 50.000 millones. Este crecimiento es debido a la popularización de placas de hardware libre, el abaratamiento de los sensores, mejora de las comunicaciones y el crecimiento de las plataformas de Internet of Things (IoT).

Nuestro objetivo es la construcción de un proyecto IoT capaz de enlazar sistemas de geolocalización, plataformas Cloud, datos en tiempo real y dispositivos conectados a internet.

Para construir un proyecto IoT, necesitamos **dispositivos**, que durante este desarrollo serán sistemas embebidos y dispositivos móviles. También necesitaremos una **plataforma** capaz de contener y gestionar toda la lógica de la información, que además proporcione la gestión de los dispositivos y su software. Como plataforma utilizaremos la PaaS Bluemix de IBM, la cual nos proporciona todas las características mencionadas anteriormente. Aquí es donde aparece el concepto de **Cloud Computing**, ya que las PaaS funcionan sobre entornos Cloud. Además, necesitaremos un **frontend**, que proporcionará la interfaz de usuario para visualizar los datos generados por la plataforma, y para ello, desarrollaremos una aplicación Android para dispositivos móviles.

La aplicación desarrollada mostrará datos de interés en función de la geolocalización del usuario. Destacamos que utilizaremos datos en tiempo real durante el desarrollo, como por ejemplo, datos de la localización espacial de aviones en un determinado espacio aéreo, o datos sobre incidencias de tráfico.

Durante este trabajo, estudiaremos la PaaS Bluemix y todas las posibilidades que nos puede ofrecer, explicaremos todos los procedimientos seguidos, las herramientas utilizadas y toda la información requerida para que el desarrollo del proyecto se realice de manera satisfactoria. Cabe destacar que este proyecto se está realizando gracias a la Cátedra UAM-IBM, y gracias a este convenio hemos extendido el período de prueba de la plataforma Bluemix y también no se nos ha cobrado ningún cargo por utilizar sus servicios.

Palabras Clave

- Internet of Things
- Cloud Computing
- Bluemix
- Platform as a Service
- RaspberryPi
- Mosquitto
- Geospatial Analytics
- Node-Red
- NodeJS
- DevOps
- Android Devices
- Real-Time data

Abstract

Nowadays we are experiencing a transformation in which devices are communicating with another devices, and such communications are creating new bussiness models, products and companies. Approximately 20 years ago, we use Internet as a information researcher. For the past 10 years, this paradigm has gone into the social, transactional and mobile field. It is estimated that the number of Internet connected devices in the year 2020 will be 50 billions. The popularizing of free hardware shields, cheaper sensors, the improving of communication and the growth of Internet of Things(IoT) platforms are responsible for this growth.

Our main goal is to build an IoT project able to link geolocation systems, Cloud computing, real-time data and internet connected devices.

To build an IoT project, we need devices which during this development will be embedded systems and mobile devices. We will need also a platform able to contain and manage the whole logic information, which also provide the management devices and their software. We will use the IBM Platform as a Service, Bluemix, which provides all the features we mentioned earlier. This is where appears the Cloud Computing concept, despite the fact that the Paas runs on Cloud environments. In addition, we need a frontend, which will provide the user interface to display data generated by the platform, and therefore develop an application for Android mobile devices.

Mobile application shows interesting data based on users geolocation. We use real-time data during this development, for example, data from aircrafts geolocation or traffic incidents.

During this work, we will study Bluemix PaaS and all the possibilities that it can offer us, we will explain all used procedures, the tools we used and all the information required for the project development will be carried out in a satisfactory way. It should be noted that this project is taking place thanks to the Cátedra UAM-IBM, and thanks to this agreement between UAM and IBM, we have extended the testing period of the platform and Bluemix has claimed any charge to use their services.

Keywords

- Internet of Things
- Cloud Computing
- Bluemix
- Platform as a Service
- RaspberryPi
- Mosquitto
- Geospatial Analytics
- Node-Red
- NodeJS
- DevOps
- Android Devices
- Real-Time data

Agradecimientos

En primer lugar quiero agradecer a Dios por haber concluido esta etapa de mi vida. Quiero agradecer a mis padres y hermanos por el apoyo que me han dado durante este periodo siendo que he pasado por momentos muy buenos y momentos no muy favorables. Quiero agradecer a todos mis amigos y compañeros con los que he compartido en esta carrera, ya que gracias a ellos he podido aprender mucho. Finalmente agradezco a la Cátedra UAM-IBM juntamente con mis compañeros de trabajo a los que quiero nombrar: David Torres, Miguel Ángel González, Ángel Lemonche, Lidia Paris, Rodrigo Amaducci, Alejando Aguirre y Borja Gil Pérez. Finalmente agradecer a Francisco Javier Gómez Arribas por sus consejos, ayuda y motivación durante el desarrollo de este trabajo.

Índice general

Índice de Figuras

Índice de Tablas

1 Introducción

1.1. Motivación del proyecto

Esta trabajo pretende utilizar las plataformas como servicio (PaaS) para gestionar las cantidades ingentes de datos del mundo de Internet de las cosas (Internet of Things). Durante el desarrollo de esta memoria utilizaremos datos de geolocalización y sensores.

Todo comenzó con la idea de gestionar grandes cantidades de datos en un servidor *cloud* relacionados con rendimientos de máquinas. Sin embargo un nuevo concepto apareció durante las investigaciones, *Internet of Things*. A grandes rasgos, internet de las cosas abarca el campo de todos los dispositivos conectados a internet como pueden ser los dispositivos móviles, automóviles inteligentes, wearables, dispositivos electrónicos, sensores; en resumen, todos aquellos dispositivos que tengan conexión a internet con capacidad de enviar y/o recibir datos. Posteriormente, el tema de análisis de grandes cantidades de datos generados por máquinas, se ha ido cambiando por grandes cantidades de datos generados por sensores, pero ahora nos encontramos con la siguiente pregunta: ¿De dónde obtenemos grandes cantidades de datos a partir de sensores? Hemos llegado a la conclusión que las API's de datos abiertos en tiempo real nos serían muy útil para este tipo de problema ya que están constantemente generando datos, y nuestra aplicación será la encargada de realizar un análisis de esas grandes masas de datos. La ventaja de la plataforma *cloud* es el hecho de que es autoescalable, con lo cual, a medida que los datos y el análisis van aumentando, la aplicación se escala en función de la cantidad de datos que se estén procesando, lo que genera a su vez, un ahorro de gastos para un usuario cuya aplicación sea dependiente del número de peticiones que se hagan a su servicio, o de la cantidad de datos que se estén procesando.

Los dispositivos móviles también pertenecen al universo de internet de las cosas, con lo cual, hemos indagado en la idea de que nuestros dispositivos móviles aporten datos generados por algunos de sus sensores, tales como el acelerómetro que nos aporta datos sobre la inclinación del dispositivos, la velocidad, altura; el sensor de luz el cual nos aporta información sobre la luminosidad ambiente o por ejemplo el dispositivo gps que todos los dispositivos móviles llevan integrados, el cual proporciona información sobre la geolocalización del dispositivo. Finalmente nos hemos decantado por el sensor gps, el cual interactuará con los datos que obtengamos de otro servidor, y de esta manera, juntar información no estructurada para generar de manera inteligente, una información útil al usuario final.

En conclusión, la motivación de este trabajo de fin de grado es utilizar una plataforma *cloud* para desarrollar una aplicación principal dividida en varios módulos integrados entre sí, con un objetivo de generar información útil a un usuario.

1.2. Objetivos y enfoque

El objetivo principal es el análisis, utilizando servicios desarrollados en plataformas Cloud, de datos proporcionados fuentes de datos o sensores conectados a sistemas empotrados, con el fin de generar información útil a un usuario final. Analizaremos las ventajas que nos proporciona cada uno de ellos comparando su rendimiento en distintos escenarios.

Para alcanzar el objetivo principal, debemos realizar los siguientes objetivos secundarios:

- Realizar un estudio sobre la plataforma como servicio Bluemix en cuanto al área de internet de las cosas y servicios de geolocalización. En cuanto a este aspecto, analizaremos las ventajas e inconvenientes de realizar una aplicación sobre una plataforma *cloud*, comparando con una aplicación desplegada en un entorno local.
- Realizar un estudio sobre protocolos de envíos de mensajes ligeros, ya que estamos enviando y recibiendo constantemente y necesitamos un protocolo de comunicación con bajo consumo de ancho de banda. Investigar los posibles protocolos de paso de mensajes y realizar una comparación entre dichos protocolos para su posterior selección.
- Realizar un estudio sobre el funcionamiento e integración de dispositivos móviles con las plataformas *cloud*. Investigar la mejor manera de comunicación entre el dispositivo y la plataforma de modo que el rendimiento de la aplicación sea óptimo.
- Realizar un estudio sobre servicios de geolocalización ya que serán necesarios para el desarrollo de nuestro servicio. Contamos con la ventaja que nuestra plataforma *cloud* cuenta con un servicio de geolocalización, con lo cual, la mejor solución para la integración de dicho servicio en nuestra aplicación, será el estudio del servicio de geolocalización que ofrece IBM en su plataforma.
- Realizar una búsqueda de portales de datos abiertos en España. A través de las APIs, obtendremos los datos de la aplicación. De igual manera realizaremos pruebas con una *RaspberryPi* de forma que sea una fuente de datos a través de sensores.
- Realizar un estudio sobre *Internet of Things*. Analizar todas las posibilidades que nos puede proporcionar el universo de IOT, y explicar su uso en el desarrollo de la aplicación. Normalmente asociamos el término Internet of Things con todos aquellos componentes electrónicos capaces de enviar datos a un dispositivo, como por ejemplo, los sensores de temperatura, sensores de luz, sensores de ultrasonido o sensores de movimiento, sin embargo, observaremos que abarca mucha más información.

El enfoque principal de la aplicación es la detección de elementos dentro de un área geolocalizable, y cuyos elementos pueden proceder de una fuente de datos tal como puede ser una API o cualquier dispositivo capaz de enviar sus coordenadas geográficas a través de internet. A partir de ese objetivo, podremos estimar el coste de la aplicación para un solo dispositivo y posteriormente simular la situación en la que muchos dispositivos realicen solicitudes al servicio desplegado en la plataforma *cloud*.

Para alcanzar dichos objetivos debemos realizar una búsqueda bibliográfica sobre la plataforma *cloud* Bluemix, *Internet of Things*, sistemas de geolocalización, dispositivos móviles y protocolos de comunicación entre componentes tanto reales como virtuales.

1.3. Metodología y plan de trabajo

La metodología debe tener en cuenta que se pretende un diseño modular. Consiste en definir los módulos que se quieren realizar para posteriormente pasar al desarrollo como pequeños ejemplos parciales, los cuales nos servirán para comprender el funcionamiento de las herramientas para cada módulo.

En vista a los conocimientos que requieren el uso de plataformas como servicio, en primer lugar procederemos a familiarizarnos con esta clase de entornos de desarrollo. Concretamente, realizaremos una lectura sobre plataformas como servicio, las posibilidades que ofrecen y las ventajas que tendremos al realizar el desarrollo sobre este tipo de plataformas. La plataforma *cloud* que hemos elegido para el desarrollo del trabajo de fin de carrera es la plataforma Bluemix de IBM, sin embargo, realizaremos una comparación entre plataformas como servicio gratuitas y de pago. Una de las más importantes es la plataforma como servicio Azure[1] de Microsoft.

En relación al protocolo de comunicación entre dispositivos, realizaremos una investigación sobre las principales formas de comunicación entre componentes, comparando la latencia, el tamaño de mensaje y el impacto que pueden provocar sobre el ancho de banda de la red. Uno de los protocolos más conocidos es el protocolo *http*, sin embargo en las secciones posteriores veremos que este protocolo no nos será útil debido a su tamaño de cabecera, y exploraremos otras alternativas más ligeras.

Para el desarrollo de la aplicación, necesitamos una fuente de datos que nos proporcione información en tiempo real en cuyo contenido tengamos datos sobre localización espacial. Para ello hemos hecho uso de API's que nos suministren dicha información. La primera fuente de datos es procedente de la web flightradar24[2] la cual nos comunica la información de los aviones de un área geográfica determinada, y entre esta información se encuentra la localización espacial de los aviones. Otra fuente de datos interesante procede del portal de datos abiertos de Madrid[3]. La aplicación solicitará información a una de las API's anteriores y generará mediante los demás módulos una información útil y comprensible al usuario final.

La aplicación será construida sobre el módulo Node-RED de *Internet of Things* de la plataforma Bluemix. En este módulo controlaremos el flujo de información a través de nodos, los cuales detallaremos en secciones posteriores. Para esta sección será necesario el desarrollo utilizando el lenguaje de programación interpretado *JavaScript*. Con dichos nodos realizaremos sencillas pruebas modulares en cuanto a filtrado de cadenas y objetos JSON, envío y recepción de información, peticiones a servidor y despliegue de servicios de geolocalización utilizando este módulo. Para ellos realizaremos programas de pruebas externos a la aplicación para comprender el funcionamiento de los nodos que están predefinidos, y el comportamiento de los nodos que definamos como desarrolladores.

Tal como hemos comentado respecto al ámbito de *Internet of Things*, hemos utilizado un dispositivo móvil debido a que contiene un sensor *gps* capaz de enviar su localización espacial. Para este caso realizaremos pruebas con aplicación sencilla capaz de mostrar la localización del dispositivo, y una vez realizada esta prueba satisfactoriamente, pasaremos a enviar dicha información a través de alguno de los protocolos de comunicación estudiados. A su vez, el dispositivo será el punto final de la aplicación, ya que el usuario recibirá la información comprensible y procesada en su dispositivo móvil. Desarrollaremos el prototipo para finalmente realizar su validación.

[1] `https://azure.microsoft.com/es-es/`
[2] `https://www.flightradar24.com/`
[3] `http://datos.madrid.es/`

1.4. Organización de la memoria

En el capítulo 2, estado del arte, se estudiará el contexto actual de las técnicas y herramientas que se emplearán en el desarrollo del trabajo.

En el capítulo 3, sistema, diseño y desarrollo, se explicará cada uno de los módulos de la aplicación, los pasos que se han seguido para su implementación y las herramientas y entornos utilizados.

En el capítulo 4, experimentos realizados y resultados, se explicará el funcionamiento de las aplicaciones desarrolladas.

En el capítulo 5, conclusiones y trabajo futuro, se sintetizarán los resultados del trabajo y se analizará si se han cumplido los objetivos propuestos. Asimismo, se plantearán los siguientes pasos del desarrollo que debe seguir el proyecto.

2

Estado del arte

2.1. Internet of Things

2.1.1. Origen y Bases

Las aplicaciones de IoT (Internet of Things) se han propagado de manera considerable en el sector industrial y su tecnología se ha utilizado durante décadas. Una de las características comunes de IoT es que los objetos deben estar "instrumentados", interconectados y ser procesados de manera inteligente en cualquier lugar, cualquier momento, de cualquier forma y cualquier modo. (Esto se conoce como las 3I [instrumented, interconnected, intelligently] y las 5A [anything, anywhere, anytime, anyway, and anyhow]).

La mayoría de las aplicaciones verticales de internet de las cosas utilizan normalmente tecnologías del nivel de red y una plataforma middleware del nivel aplicación tales como redes cableadas o inalámbricas estándares, DBMS, frameworks de seguridad, webs basadas en middlewares de nivel 3, PaaS (platform as a service) multiusuario, interfaces SOA (service-oriented architecture), etcétera. Todas estas tecnologías comunes se pueden consolidar con un propósito general, un framework y una plataforma escalable para proporcionar un mejor servicio a las aplicaciones verticales.

Los SMP's (Service Managemente Platforms) son la clave para entrar al mercado "machine to machine". A través de ellos se permiten la gestión esencial de la conectividad, manejo inteligente del plan económico y la capacidad de proporcionar al cliente un autoservicio los cuales son requisitos fundamentales hoy en día para proveer de manera eficiente un servicio M2M (machine to machine).

Otra característica común que IoT ha proporcionado al mundo de las TIC es el cambio significativo sobre la forma en la que se genera la información, proveniente de una generación masiva de datos de las máquinas sin intervención humana. [1]

2.1.2. Ideas

Internet de las cosas va asociada a sensores. En primer lugar debemos realizar un estudio de todos los posibles sensores que encontremos para barajar las posibilidades con las que nos

encontramos.

Los cuatro pilares de internet de las cosas son M2M, RFID, WSN's (Wireless sensor networks) y SCADA (supervisory control and data acquisition).

- M2M (machine to machine) utiliza dispositivos para capturar eventos desde conexiones de redes híbridas (inalámbricas o cableadas) a un servidor central (software) que traduce los eventos capturados a información con sentido (informa sobre errores para que sean corregidos)
- RFID utiliza ondas de radio para trasferir datos desde un dispositivo electrónico a un sistema central con el propósito de identificación y seguimiento del objeto.
- WSN (Wireless Sensors Network) consiste en una distribución espacial de sensores que monitorizan condiciones físicas y medioambientales, y transfieren la información obtenida de los sensores que trabajan de manera cooperativa a través de una red normalmente cableada o híbrida, a un lugar principal.
- SCADA es un sistema autónomo basado en la teoría de control de bucle cerrado, o un sistema inteligente, o un CPS que conecta monitores y controla equipos a través de la red.

Podemos ver las categorías de los cuatro pilares y las tecnologías de redes diferentes en función de su relevancia en el cuadro 2.1 (Las letras S y L significan Short y Large consecutivamente).

Pilares	S-Range Wireless	L-Range Wireless	S-Range Wired	L-Range Wired
RFID	Alta	Baja	Nula	Baja
WSN	Alta	Baja	Nula	Baja
M2M	Baja	Alta	Nula	Baja
SCADA	Baja	Alta	Alta	Alta

Cuadro 2.1: Relevancia de los pilares en función de la tecnología de red

Estos son los cuatro pilares de internet de las cosas, e internet de las cosas es el pegamento que uno estos cuatro tipos de redes siguiente una metodología de red común y una plataforma middleware (ej. Bluemix). Todo ello permite al usuario conectar todos sus dispositivos físicos con una infraestructura común y una metodología consistente para leer los datos procesados por las máquinas e interpretarlos para mostrar al usuario qué está sucediendo.

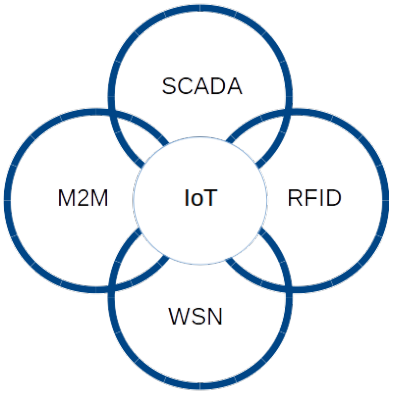

Figura 2.1: Cuatro pilares de Internet of Things

2.1.3. RaspberryPi

RaspberryPi es un pequeño ordenador con arquitectura ARM. Este pequeño ordenador se ha diseñado sobre una placa de montaje como un proyecto de universidad. ARM es una arquitectura RISC (Reduced Instruction Set Computer) de 32 o 64 bits. Un enfoque de diseño basado en RISC, causa que los procesadores ARM necesitan una cantidad menor de transistores que los procesadores x86 CISC típicos en la mayoría de ordenadores personales. Este enfoque de diseño por lo tanto nos lleva a una reducción de los costes, calor y energía. Estas características son deseables para dispositivos que funcionan con baterías, como teléfonos móviles, tablets u otros dispositivos electrónicos.

En cuanto al sistema operativo funcional en una RaspberryPi, pueden correr sistemas Linux que soporten procesadores ARM. Ahora mismo la distribución más estable y cómoda para el usuario es Raspbian, pero existen otra como Pidora o ArchLinux ARM. El sistema operativo se instala sobre una tarjeta SD de al menos ocho gigabytes de memoria.

Las características del último modelo de las RaspberryPi es la versión 3 modelo B, y posee las siguientes características:

- 1.2GHz 64-bit quad-core ARMv8 CPU
- 802.11n Wireless LAN
- Bluetooth 4.1
- Bluetooth Low Energy (BLE)
- 1GB RAM
- 4 USB ports
- 40 GPIO pins
- Full HDMI port
- Ethernet port
- Combined 3.5mm audio jack and composite video
- Camera interface (CSI)

- Display interface (DSI)
- Micro SD card slot (now push-pull rather than push-push)
- VideoCore IV 3D graphics core

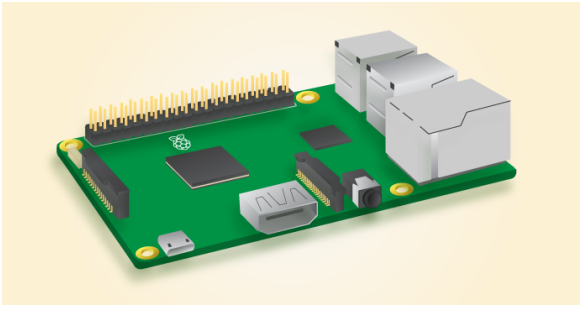

Figura 2.2: RaspberryPi 3 Model B

Para poder utilizar la RaspberryPi como un servidor, tendremos que asegurarnos que esta tenga una dirección IP estática, ya que el servicio debe saber en todo momento la dirección IP de nuestra sistema embebido. El sistema operativo, Raspbian, basado en Linux, nos permite acceder fácilmente al dispositivo

2.1.4. Dispositivos Android e IoT

Android ha creado una red de dispositivos conectados entre sí por la que fluye una gran cantidad de datos que pueden ser interesantes al mundo de Internet of Things. Gracias al sistema operativo Android, no solamente se ha logrado una hacer realidad el concepto de Internet of Things, sino que se ha conseguido que sea útil.

Una de las razones por las que Android está relacionado con Internet of Things es el hecho de que cualquier dispositivo con el sistema Android, puede actuar como una aplicación front-end para cualquier dispositivo o sensor capaz de emitir datos a través de internet mediante protocolos de comunicación.

La segunda razón consiste en que las aplicaciones Android obtendrán un mayor provecho de los dispositivos electrónicos de IoT. En cuanto al desarrollo de la app Android, no se requiere un nivel avanzado de programación para desarrollar un front-end capaz de emitir órdenes a aquellos dispositivo capaces de reaccionar a una acción, o recibir datos de aquellos dispositivos que están constantemente enviando datos al dispositivo móvil, tal como puede ser un sensor de temperatura.

Una tercera razón es el hecho de que Java está muy bien integrado con protocolos de comunicación de IoT, y las aplicaciones Android se desarrollan en Java, que facilita su portabilidad. Sin embargo la compatibilidad de los protocolos de comunicación con otros lenguajes está avanzando rápidamente con lo cual, ese aspecto no es un problema futuro.

2.1.5. Protocolo MQTT

MQTT son las siglas de Message Queue Telemetry Transport y tras ellas se encuentra un protocolo diseñado por IBM para la comunicación Machine-to-Machine (M2M). Está enfocado en el envío de datos en aplicaciones donde se requiere muy poco ancho de banda. Además, sus requisitos para funcionar son muy bajos. Estas características han hecho que rápidamente se

convirtiera en el protocolo más empleado en la comunicación de sensores, consecuentemente, dentro del mundo de IoT.

En cuanto a su arquitectura, sigue una topología en estrella, donde existe un nodo central o broker, con capacidad para trabajar con un máximo de 10.000 clientes. El broker es el encargado de gestionar la red y transmitir los mensajes. Una característica interesante es la capacidad de MQTT para establecer comunicaciones cifradas, lo que aporta a nuestra red una capa extra de seguridad.

Dentro de la arquitectura MQTT, el concepto *topic* es esencial ya que a través de ellos se articula la comunicación puesto que emisores y receptores deben estar suscritos a un *topic* común para poder establecer la comunicación. Este concepto es prácticamente el mismo que se emplea en colas, donde existen publicadores y subscriptores, como por ejemplo Twitter. Además, los *topics* tienen una estructura jerárquica gracias a la cual podemos establecer relaciones padre-hijo de forma que si nos suscribimos a un *topic* padre, recibiremos información de sus correspondientes hijos. Las características más importantes de MQTT son:

- Experiencia de usuario: la información se ejecuta en tiempo real, el uso de batería es muy bajo.
- Flexibilidad y escalabilidad: Un simple servidor puede gestionar millones de usuarios y dispositivos.
- Bajos costes de ejecución: Compacto y bajos costes de red.
- Bajos costes de desarrollo: Sencillo de desarrollar y se integra fácilmente a cualquier sistema.

Podemos compararlo con http en el que un mensaje vacío ocupa 302 bytes mientras que un mensaje MQTT vacío ocupa 69 bytes. Si hablamos de grandes cantidades de datos, obtendremos una optimización del ancho de banda muy considerable.

2.2. Cloud Computing

2.2.1. Introducción

En las últimas décadas los procesos de deslocalización e internacionalización de las grandes empresas, unidos a la explosión en el uso la de tecnologías de información y procesamiento de datos, han hecho que las necesidades de cómputo de las grandes empresas y organizaciones hayan crecido a un ritmo superior al que lo hacía la capacidad de cálculo de los ordenadores personales. Por este motivo, y para satisfacer las necesidades de los sistemas de computación más exigentes, se ha producido una interesante evolución de las arquitecturas de cálculo, basada fundamentalmente en la ejecución simultánea de procesos en múltiples equipos informáticos[2].

Origen de Cloud Computing

Hace muchos años ya se introdujeron conceptos de Cloud Computing esto por John McCarthy en 1960, quien dijo en un discurso que "Algún día la computación podrá ser organizada como un servicio público" y haciendo una comparación con lo ocurrido en la distribución eléctrica a comienzos del siglo XX donde las grandes empresas tenían su propio generador de energía, ahora las empresas grandes constan de sus servidores, clúster o supercomputadoras[3].

Debido a las necesidades de cómputo descritas anteriormente, se ha estado realizando un importante esfuerzo en la investigación de capacidades en la ejecución de procesos en múltiples ordenadores. Esta tendencia fue impulsada originalmente por la utilización de sistemas abiertos, interoperables y protocolos de comunicación estándar los cuales permitían la comunicación eficiente entre sistemas y tecnologías heterogéneas.

El primer paso de esta tecnología fue propiciado en gran manera por los sistemas Unix que permitieron la configuración de clusters, es decir, agrupaciones de ordenadores con componentes hardware comunes que se comportan como un único ordenador.

Tras varias décadas de investigaciones y desarrollos en estas tecnologías, la aparición del sistema Linux con sus estándares abiertos permitieron crear clusters basados en las arquitecturas PC, consiguiendo instalaciones de alto rendimiento popularizando esta solución durante la década de 1990[4].

Estos clusters sufrieron un proceso de especialización para proporcionar servicios de cálculo y almacenamiento, fundamentalmente en centros de investigación y universidades. Estos centros comenzaron a ofrecer sus servicios a tercero a través de protocolos estándar, constituyendo la denominada arquitectura de computación GRID, orientada al procesamiento en paralelo o al almacenamiento de gran cantidad de información.

Estas arquitecturas fueron acogidas en instituciones investigadoras durante la primera mitad de la década de 2000, pero la complejidad para utilizar la infraestructura, las dificultades para utilizar los diferentes sistemas GRID, y los problemas de portabilidad entre ellas, hicieron que nunca se popularizara fuera del ámbito de la investigación y académico.

Fue en ese entonces cuando empezaron a popularizarse las tecnologías de virtualización que hacían posible desacoplar el hardware del software, y permiten replicar el entorno del usuario sin tener que instalar o configurar todo el software que requiere cada aplicación. Este aspecto tiene muchas ventajas en la distribución y el mantenimiento de sistemas de software complejos y permite integrar bajo un mismo entorno un conjunto de sistemas heterogéneos.

Esta nueva arquitectura permitía distribuir la carga de manera mucho más sencilla, lo cual eliminaba los problemas que presentaba la arquitectura GRID, abriendo una nueva puerta al cálculo distribuido, llamado *Cloud Computing*. En la figura 2.3 podemos observar la evolución de las arquitecturas de cálculo a lo largo del tiempo.

Concepto de Cloud Computing

Atendiendo a la definición dada por el NIST[5] (National Institute of Standards and Technology), el *cloud computing* es un modelo tecnológico que permite el acceso ubicuo, adaptado y bajo demanda en red a un conjunto compartido de recursos de computación configurables compartidos (por ejemplo: redes, servidores, equipos de almacenamiento, aplicaciones y servicios), que pueden ser rápidamente aprovisionados y liberados con un esfuerzo de gestión reducido o interacción mínima con el proveedor del servicio.

Otra definición complementaria es la aportada por el RAD Lab[6] de la Universidad de Berkeley, desde donde se explica que el *cloud computing* se refiere tanto a las aplicaciones entregadas como servicio a través de Internet, como el hardware y el software de los centros de datos que proporcionan estos servicios. Los servicios anteriores han sido conocidos durante mucho tiempo como Software as a Service(SaaS), mientras que el hardware y software del centro de datos es a lo que se llama nube.

Con esta información sintetizada en este apartado, observamos que el *cloud computing* supone un cambio significativo respecto a como procesar la información y como gestionar las áreas TIC. Además, en el modelo antiguo, necesitábamos realizar inversiones en cuanto a hardware, software,

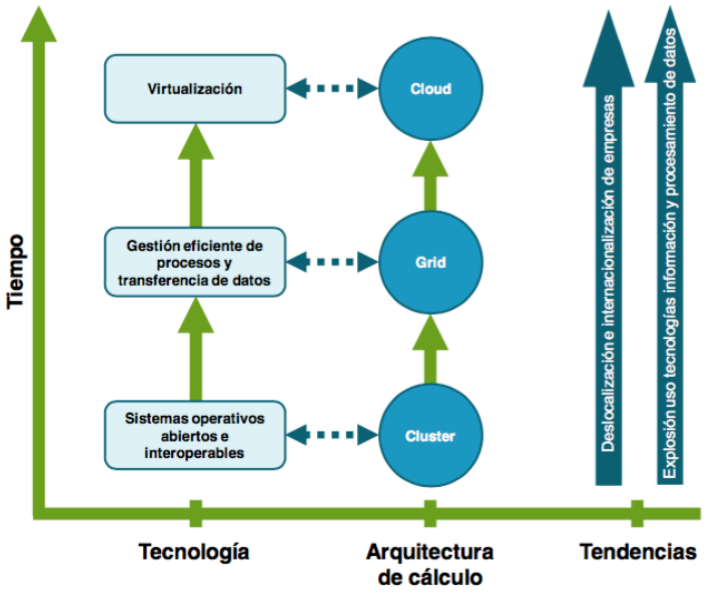

Figura 2.3: Evolución de las arquitecturas de cálculo

centros de procesamiento de datos, redes, personal o seguridad, mientras que en el modelo *cloud* eliminamos la necesidad de grandes inversiones iniciales y costes fijos.

Características de Cloud Computing

Para poder entender de una manera rápida y sencilla cuales son las claves[7] del concepto del *cloud computing*, se recurre a una serie de características principales que lo diferencian de los sistemas tradicionales de explotación de las TIC. Entre las características asociadas al *cloud computing* se encuentran las siguientes:

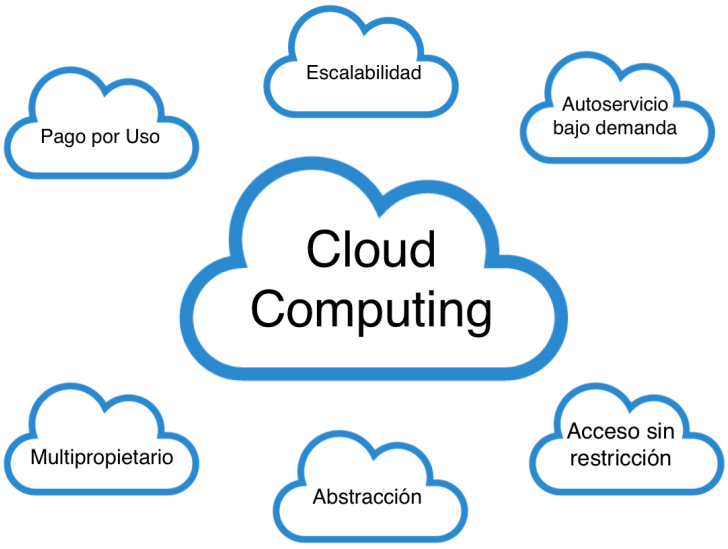

Figura 2.4: Características de *cloud computing*

- **Pago por uso**
  Una de las características principales de las soluciones cloud es el modelo de facturación basado en consumo. Se realiza el pago en función del uso que se realiza del servicio cloud.

- **Abstracción**
  Capacidad de aislar los recursos informáticos contratados del proveedor de servicios cloud de los equipos informáticos de cliente. Esto se consigue gracias a la virtualización, con lo cual, no necesitamos estar pendientes del mantenimiento de la infraestructura, actualización de sistemas o realización de pruebas.

- **Agilidad en la escalabilidad**
  Característica o capacidad consistente en aumentar o disminuir las funcionalidades ofrecidas al cliente, en función de sus necesidades puntuales sin necesidad de nuevos contratos ni penalizaciones. De la misma manera, el coste del servicio asociado se modifica también en función de las necesidades puntuales de uso de la solución. Esta característica, relacionada con el pago por uso, evita los riesgos inherentes de un posible mal dimensionamiento inicial en el consumo o en la necesidad de recursos.

- **Multiusuario**
  Capacidad que otorga el cloud que permite a varios usuarios compartir los medios y recursos informáticos, permitiendo la optimización de su uso.

- **Autoservicio bajo demanda**
  Esta característica nos permite acceder de manera flexible a las capacidades de computación en la nube de forma automática a medida que las vaya requiriendo, sin necesidad de una interacción humana con su proveedor o proveedores de servicios cloud.

- **Acceso sin restricciones**
  Característica consistente en la posibilidad ofrecida a los usuarios de acceder a los servicios contratados de *cloud computing* en cualquier lugar, en cualquier momento y con cualquier dispositivo que disponga de conexión a redes de servicio IP. El acceso a los servicios de *cloud computing* se realiza a través de la red, lo que facilita que distintos dispositivos, tales como teléfonos móviles, dispositivos PDA u ordenadores portátiles, puedan acceder a un mismo servicio ofrecido en la red mediante mecanismos de acceso comunes.

2.2.2. Tecnologías asociadas al Cloud

Este nuevo modelo de computación y de negocio requiere de una tecnología similar a la arquitectura Grid, que soporte las necesidades de recursos hardware que presenta: por otro lado son necesarias tecnologías a un nivel de abstracción más alto que permitan ofrecer estos recursos a los usuarios, En concreto estas tecnologías son la virtualización y los servicios web, que resuelven la necesidad de ofrecer recursos hardware y software de distintos tipos siguiendo unos estándares, a través de la red, con la propiedad de multi-tenancy o compartición por parte de varios usuarios y manteniendo siempre la transparencia deseada[8].

Sistemas Grid

Las arquitecturas Grid[9] surgieron como la evolución natural de los clusters, generalizando el concepto gracias a la evolución de Internet. Aunque ambos tipos de arquitectura son de memoria distribuida (es decir, cada uno de los nodos que componen el sistema tiene su propia memoria principal que no es accesible para el resto de nodos, por lo que la colaboración y comunicación entre ellos debe realizarse de manera explícita mediante paso de mensajes) existen importantes diferencias entre ellas que se pueden resumir en:

- En el caso de un cluster, los nodos están ubicados geográficamente en el mismo lugar y se conectan entre ellos mediante una red controlada de altas prestaciones mientras que en el caso de un Grid, los nodos están distribuidos geográficamente e interconectados mediante una red de área extensa. Con los grids se comienza a hablar de arquitecturas débilmente acopladas, es decir, compuestas por nodos de cómputo distribuidos con mucha independencia entre ellos y poco grado de interacción, que trabajan de manera conjunta pero no son realmente una única arquitectura. En este tipo de sistemas la interacción entre los nodos se produce mediante el paso de mensajes.
- Mientras que en un cluster todos los nodos que componen el sistema son propiedad de una misma organización, en el caso de un Grid, los nodos suelen tener diferentes propietarios que colaboran compartiendo sus recursos para formar o bien un grid de datos o bien uno de cómputo.

Servicios Web

Como la infraestructura tecnológica que sustenta el modelo Cloud es un grid, parece razonable pensar que la utilización de servicios web[10] a para resolver muchos de los problemas detectados facilitará la evolución del nuevo paradigma. De hecho, con el paso del tiempo se ha comprobado que las arquitecturas orientadas a servicios son muy adecuadas para explotar sistemas heterogéneos y débilmente acoplados. Service Oriented Aarchitecture (SOA) se puede comprender como una filosofía de diseño de aplicaciones que propone una alternativa a las tradicionales aplicaciones de negocio monolíticas basadas en componentes y objetos.

Habitualmente, las empresas compran sus aplicaciones de negocio a proveedores de software o realizan desarrollos a medida. Ambas soluciones pueden llegar a ser muy costosas, ya que no permiten la reutilización de código y además no siempre evolucionan al mismo ritmo que la organización, ya que su actualización, escalado y mejora suponen grandes inversiones.

SOA propone que los procesos de negocio no se traduzcan en aplicaciones monolíticas clásicas con necesidad de multitud de interdependencias entre ellas, sino que éstos llamen o invoquen a los servicios que necesitan para obtener resultados. Estos servicios deben estar débilmente acoplados e interoperables, por lo que son muy importantes los protocolos que definen formalmente estos servicios y que permiten la comunicación con ellos y entre ellos. En cualquier caso, la reutilización de código es esencial, al igual que la flexibilidad del software programado con esta filosofía y sus posibilidades de actualización y escalado.

En los últimos tiempos han surgido diferentes especificaciones que pretenden ser una alternativa a esta pila de protocolos estándar en su nivel más bajo. Por ejemplo, se ha extendido mucho el protocolo REST (Representational State Transfer), que permite que los servicios que se comuniquen entre sí con XML directamente sobre HTTP evitando así la complejidad que introducen las cabeceras y metadatos que SOAP añade a XML. Y también se encuentran cada vez más implementaciones que utilizan JSON para evitar la utilización de XML.

Virtualización

Las técnicas de virtualización se conocen y se utilizan desde hace más de 30 años para implementar recursos informáticos aislando unas capas del sistema de otras: hardware, sistema operativos, aplicaciones, datos, redes. La idea es que si conseguimos este aislamiento, el alcance de cualquier cambio o modificación en una de las capas es mucho más restringido que en los sistemas tradicionales y casi no afecta a las demás. Para conseguirlo es necesario que cada capa tenga unos interfaces estándar con el exterior, de manera que desde fuera siempre se vea de la misma forma.

Se pueden distinguir distintos tipos de virtualización en función de la capa que se abstraiga. Quizás una de las más conocidas sea la virtualización del hardware. En estos casos se utiliza un *Hypervisor* para gestionar hardware subyacente. Esto permite ejecutar más de un sistema operativo en un único equipo físico sin necesidad de tener diferentes particiones.

Es posible virtualizar aplicaciones para separar la aplicación del sistema operativo sobre el que está instalada, reduciendo así los conflictos entre aplicaciones y simplificando las distribuciones y actualizaciones de software. En el caso de Cloud Computing, la virtualización de máquina permite que los proveedores ofrezcan una infraestructura deslocalizada a sus clientes, de manera que para ellos la ubicación real de sus recursos sea completamente transparente, y permite también que varias máquinas virtuales asociadas a diferentes usuarios compartan recursos hardware en un único equipo físico. El resto de tipos de virtualización completan en muchos casos a esta, que es la más básica pero también la imprescindible, ya que se necesita para independizar las aplicaciones distribuidas de los recursos hardware, y conseguir el dinamismo y la flexibilidad deseada.

2.2.3. Ventajas

Reducción de gastos

El modelo de facturación *pago por uso* utilizado en la tecnología cloud supone una considerable reducción de los gastos generales, tales como los derivados de actualizaciones de software, servidores para el almacenamiento de datos, mantenimiento, etc. Los servicios basados en la Nube se pagan en función del consumo y, por tanto, el precio dependerá de las necesidades de cada empresa y de la utilización que cada una realice de sus servicios. De este modo, la inversión inicial es mucho menor y los costes generales se reducen significativamente con respecto al sistema tradicional.

Flexibilidad

Satisfacer cualquier demanda de forma casi inmediata es posible gracias a la gran capacidad de los servidores remotos en los que se encuentran alojados los servicios. La flexibilidad en este sentido es vital y, sin duda, la rapidez de la respuesta que proporciona esta tecnología es uno de los motivos más valorados y determinantes para la adopción de *cloud computing*.

Recuperación de datos

En cuanto la la pérdida de información,*cloud computing* permite mantener una copia exacta del servidor, para solucionar con rapidez y de forma eficaz un problema inminente relacionado con pérdida de datos importantes para la empresa o el usuario.

Rápida implementación

La sencillez y simplicidad de esta tecnología es una de sus mayores ventajas. La única condición indispensable es la de disponer de conexión a Internet, no necesitamos requisitos hardware/software para dar inicio al desarrollo. Por tanto, la implementación es rápida, sencilla y ágil.

Actualizaciones automáticas

La tecnología *cloud computing* permite la actualización automática de todas las aplicaciones alojadas en el servidor conservando, eso sí, todas las personalizaciones e integraciones realizadas previamente durante la actualización. Además, los propios proveedores *cloud* se encargarán del mantenimiento del servidor y, con ello, de todas las actualizaciones que sean necesarias.

Competitividad

La tecnología cloud pone al alcance de la pequeña y mediana empresa una tecnología avanzada, que permite optimizar el trabajo, dar respuesta a las posibles demandas, optimizar sus negocios y, en definitiva, alcanzar una importante mejora competitiva en el mercado, además de una reducción de costes e inversión de forma clara y significativa. Esto aporta a las pequeñas y medianas empresas la posibilidad de poder competir con grandes empresa, que hasta ahora contaban con muchos más recursos que ellas.

Sostenibilidad y Green TIC

El uso de la tecnología cloud en lugar de las soluciones tradicionales (on-premise) permite reducir el consumo de energía y la emisión de gases contaminantes de los recursos de TI de las empresas usuarias. De acuerdo al informe "Situación y retos de las Green TIC en España"[11] presentado en Marzo de 2012 por la Asociación de Empresas del Sector TIC, las Comunicaciones y los Contenidos Digitales (Ametic), FUNCOAS (Fundación para la transferencia de conocimiento de Ametic) y la Plataforma Tecnológica GreenTIC, un entorno *cloud computing* puede suponer un ahorro energético y de espacio superior al 60 %.

2.2.4. Desventajas

Así como existe una gran cantidad de ventajas, existe un número también elevado de desventajas que deben ser tomadas en cuenta a la hora de optar por el uso de este nuevo modelo de computación[12]. La computación cloud siempre tendrá que tener en cuenta algunas características, las cuales serán a su vez su reto durante el periodo en el que esta tecnología esté vigente. Muchas de las empresas o servicios hoy en día no utilizan las plataformas como servicios, y una de las causas puede ser los riesgos que conlleva migrar a esta tecnología

Dependencia del proveedor

Todos los servicios que añadimos a nuestra aplicación, y todo el entorno sobre el que se ejecuta nuestros proyectos, son proporcionados por un proveedor de servicios. Nuestro modelo de negocio depende del proveedor, y en su caso, si no trabajamos con un proveedor fiable, podemos encontrarnos un obstáculo en nuestro desarrollo ya que experimentaremos fallos de servicios, lo que se traduce en pérdida de clientes y de capital. Cuando seleccionamos un proveedor, debemos estar muy seguros de su fiabilidad y consultar como actuarían en un posible caso de fallo de servicio.

Servicios poco personalizables

Para las pequeñas organizaciones este puede ser un punto más crítico y con mayor dificultad que los es para las grandes organizaciones, quienes cuentan con un departamento de TI con el

personal capacitado para realizar todas estas tareas de ajuste y personalización de las aplicaciones a sus necesidades. Por lo general, las aplicaciones bajo el esquema de SaaS son algo de lo que se puede disponer mas no modificar. En muchos casos las aplicaciones desarrolladas bajo demanda, a las que se tiene acceso en el modelo tradicional suelen tener una gran cantidad de funcionalidad desarrolladas específicamente para el usuario, lo cual no ocurre por lo general en la nube y esto suele ser un gran obstáculo para ser parte de la nube.

Alta latencia

Todas las aplicaciones en la nube sufren de este problema asociado a la latencia generada por las conexiones WAN (Wide Area Network) con la que el usuario se conecta a la infraestructura de la nube. Esta restricción hace que las aplicaciones con tareas de alto procesamiento de datos sean óptimas para usar este modelo, mientras que las aplicaciones que requieren de la transferencia de volúmenes de datos considerables o con modelos de transferencia de mensajes, de cualquier tamaño, entre varias unidades de procesamiento, no lo son debido a la latencia en las comunicaciones.

Sistema sin estado

Todos los sistemas en Cloud Computing no poseen la capacidad de llevar un estado de las comunicaciones, como ocurre por lo general en cualquier sistema en internet. La propia arquitectura de este tipo de infraestructura hace que las comunicaciones deban ser unidireccionales, como ocurre con todas las solicitudes HTTP que se realizan (PUT y GET), logrando que cada petición tenga su respuesta pero sin garantizar que se tenga una conversación a través de varias peticiones. Esto se debe a que cada mensaje, al ser un sistema distribuido, puede tomar rutas diferentes y no se garantiza el orden de llegada de cada mensaje, aunque debido a esta naturaleza se garantiza que todos los mensajes son entregados. Esto hace que sea necesario la implementación de encabezados y de capas intermedias (middleware) para lograr este tipo de funcionalidades.

Privacidad y seguridad

Una de las desventajas más graves que existe actualmente, al tiempo de ser el reto más grande al que se ven afrontadas las compañías, y que cualquier usuario que desee usar un sistema en la nube debe tener en cuenta es la privacidad y la seguridad de los datos[13]. Aun cuando el proveedor del servicio, a través de los acuerdo de niveles de servicio (SLA) se comprometen a llevar un control de la seguridad del aplicación y la infraestructura, así como de la privacidad de la información de la información almacenada en sus instalaciones, existe un riesgo remanente que no puede ser eliminado ni olvidado. El riego existe en que al estar la información viajando y permaneciendo en una infraestructura que no se puede controlar, se incrementa el riesgo que dicha información pueda ser interceptada o modificada por un tercero. Pero el peor problema consiste en el marco legal que involucra y que todavía no ha sido desarrollado para estos ambientes de prestación de servicios. Actualmente, aunque es posible delegar la funciones, no es posible delegar la responsabilidad de la información[14], así que ante el gobierno es la empresa la responsable de dicha información, por lo que al no tener el control de la infraestructura donde está viviendo, es decir la nube, no es posible tomar las medidas de protección o al menos no se sabe con qué medidas cuenta el proveedor para asegurar el nivel de seguridad exigido debido a la virtualización de los ambientes.

2.2.5. Modelos de servicio Cloud

Además de atender al propietario de la infraestructura tecnológica y al modelo de explotación de la misma, se pueden distinguir también entre diferentes tipos de sistemas cloud dependiendo del tipo de servicio que ofrezcan. Desde los inicios de este paradigma se ha empleado la nomenclatura XaaS para indicar que el modelo es "X as a Service". Dentro de esta forma de clasificación existen tres modelos universalmente reconocidos: IaaS, PaaS y SaaS.

Infraestructure as a Service (IaaS)

Se trata de uno de los modelos de servicio más básicos de Cloud Computing en el que los clientes pagan por el uso de diversos servicios como capacidad de procesamiento (servidores virtuales), almacenamiento (espacio en disco) o ancho de banda. Ejemplos de proveedores de IaaS incluen a IBM (Softlayer), Amazon o Rackspace.

El proveedor de tecnología ofrece a sus clientes la infraestructura tecnológica básica sobre la que trabajar, es decir, máquinas virtuales con diferentes características hardware y acceso a un interfaz o API para configurarlas y gestionarlas. También se suelen ofrecer herramientas gráficas para realizar monitorizaciones sencillas o consolas para trabajar con comandos. El cliente debe escoger el número y tipo de recursos que necesita y los utiliza como si fueran equipos o servidores físicos de sus propiedad, por tanto, debe configurarlos e instalar las aplicaciones que necesite. Normalmente los proveedores de IaaS ofrecen plantillas de máquinas virtuales con nas características determinadas y sistemas operativos básicos instalados, aunque también suele ser posible que los clientes especifiquen las características concretas de las máquinas virtuales que necesitan e instalen sus propios sistemas operativos. En este tipo de servicio se suele pagar por el número y tipo de máquina virtual contratada.

Platforms as a Service (PaaS)

Se trata de un modelo de servicio de *cloud computing* relativamente sofisticado que permite que los clientes desarrollen sus aplicaciones sobre una plataforma que corre en la nube y que es elástica. Esto significa que la aplicación se adapta automáticamente al nivel de usuarios que tiene en un momento dado sin necesidad de intervención humana para garantizar un determinado nivel de servicio. La plataforma normalmente incluye servicios como base de datos o mensajería lo que simplifica el desarrollo de aplicaciones para los programadores. Dado que la aplicación es elástica, si hay pocos usuarios el cliente pagará muy poco por el servicio y el costo irá aumentando a medida que crezca la demanda. Ejemplos de PaaS son Google App Engine, Microsoft Azure o Bluemix. Los proveedores de IaaS se pueden convertir en proveedores de PaaS utilizando una plataforma abierta como OpenStack, como lo han hecho Rackspace o IBM.

En este caso el proveedor ofrece, no solo los recursos hardware junto con un sistema operativo básico, sino también la plataforma sobre la que poder desarrollar y ejecutar distintos tipos de servicios o aplicaciones. Por lo tanto, recursos tales como librerías, entornos de desarrollo, *runtimes*, motores de bases de datos o middleware de comunicaciones. El cliente utiliza estos recursos contratados para desarrollar o instalar sus propias aplicaciones a través de las API's ofrecidas por el proveedor. Un problema que surge con este modelo, es el hecho de que la mayor parte de las API's son propietarias de los diferentes proveedores y pueden generar cautividad y dependencia del proveedor. Es decir, una aplicación o servicio desarrollado sobre cierta PaaS, es muy probable que no funcione sobre la plataforma de otro proveedor sin tener que realizar modificaciones. En este tipo de servicio se suele pagar por número de usuarios y tipo de plataforma contratada.

Software as a Service (SaaS)

Conocido generalmente como *SaaS*, es un modelo de distribución de software donde el soporte lógico y los datos que maneja se alojan en servidores de una compañía de de tecnologías de la comunicación e información (TIC), a los cuales se acceden a través de internet desde un cliente. Regularmente el software puede ser consultado en cualquier ordenador. Se deduce que la información, el procesamiento, y los resultados de la lógica de negocio del software, están hospedados en la compañía de TIC.

SaaS ofrece un servicio completo y con pocas opciones de personalización, es decir, proporciona la infraestructura, el software, la solución y toda la pila de aprovisionamiento como un servicio global. Todas los aspectos que no sean la interacción con la aplicación son trasparentes al usuario. En el modelo SaaS, los usuarios pagan por el uso de los servicios mediante suscripciones válidas por un determinado periodo de tiempo, como puede ser la aplicación Spotify, o pueden ser gratuitas como por ejemplo Gmail de Google.

2.3. Bluemix, Platform as a Service (PaaS)

2.3.1. Introducción

Bluemix es una plataforma como servicio desarrollada por IBM, en la que se pueden desplegar aplicaciones realizadas en diferentes lenguajes de programación y que pueden consumir servicios[1]. Utiliza la metodología de desarrollo DevOps de forma integrada la cual permite integrar, ejecutar, desplegar y gestionar aplicaciones en entornos *cloud*. Bluemix está basado en la tecnología de código abierto CloudFoundry y se ejecuta sobre la infraestructura SoftLayer, la infraestructura como servicio de IBM.

Una de las grandes ventajas de Bluemix es el hecho de soportar Java, NodeJS, GO, Python, Ruby Sinatra, Ruby on Rails, C, C++, php, y otros lenguajes de programación. Además puede ser extendido a otros lenguajes de programación como Scala mediante el uso de buildpacks.

La plataforma Bluemix se ha desarrollado por un equipo localizado en diversas partes del mundo. Fue abierta al público en junio de 2014. En ese momento, Bluemix era de uno de los despliegues de CloudFoundry más grandes del mundo[2].

Bluemix permite la gestión de las aplicaciones a través de un portal web o desde una herramienta cliente desde la línea de comando (cmd). Además toda la plataforma se ejecuta sobre la infraestructura como servicio (IaaS) de SoftLayer[15]. SoftLayer es un servidor dedicado, servicio de alojamiento dedicado y proveedor de *cloud computing*, fundado en 2005 y adquirido por IBM en 2013. Al tratarse de una PaaS, significa que el acceso y utilización es a nivel de middleware de ejecución de aplicación y API de acceso a cada uno de los servicios, nunca a nivel de sistema operativo.

Además una característica importante es el hecho de ser gratuita. Hay un modelo de uso limitado en capacidad y recursos gratuito. A medida que el uso de la aplicación vaya creciendo, se irán realizando pagos por consumición de los recursos. Hay muchos servicios disponibles. IBM ha incluido y sigue incluyendo servicios basados en su middleware, sin embargo se pueden añadir servicios de terceros. La idea es que todo esté integrado a nivel de tarificación. Al tratarse de una tecnología cloud, es muy rápido crear una aplicación y llevarla a producción para empezar a dar el servicio y por último es extensible. Si no existe el servicio que necesitamos, se puede construir e integrarlo en la plataforma.

[1] Catálogo de Servicios IBM Bluemix: `https://console.ng.bluemix.net/catalog/`

[2] *IBM anuncia la disponibilidad de Bluemix, su plataforma abierta de desarrollo cloud*

En las siguientes subsecciones, realizaremos una descripción detallada de cada uno de los servicios utilizadas durante el desarrollo del software desarrollado durante este período y una explicación detallada de aquellos conceptos necesarios para entender lo qué es la plataforma Bluemix.

2.3.2. CloudFoundry

Bluemix está implementado sobre CloudFoundry, así que en primer lugar debemos conocer qué es CloudFoundry y cuáles son sus principales características, ya que por herencia, Bluemix también las tendrá y observaremos lo que aporta sobre esta base.

CloudFoundry es un software libre que presenta una capa de servicio PaaS sobre un entorno IaaS apoyándose en la virtualización. Define una serie de protocolos y convenciones sobre cómo se gestiona el ciclo de vida de una aplicación, los servicios que tiene disponibles para su consumo y cómo debe adaptarse a la demanda o carga de trabajo[16].

CloudFoundry nos proporcionará una infraestructura y las herramientas para ejecutar aplicaciones, y nos permitirá gestionar los recursos que necesiten de una forma automática. Por ejemplo, una instancia de CloudFoundry te permitirá desplegar, ejecutar, y gestionar la capacidad de servicio de una o varias aplicaciones basadas en Java sin tener que lidiar con la instalación y configuración ni del sistema operativo, ni del servidor de aplicaciones, ni de la base de datos, y lo mismo con otros lenguajes y servicios, ya que no está vinculado a ningún lenguaje de programación, ni servicio concreto, ni infraestructura propietaria.

Desde el punto de vista del usuario podemos interactuar con CloudFoundry de dos maneras. En primer lugar podemos utilizar la línea de comandos. CloudFoundry dispone de un cliente/aplicación basado en línea de comandos que permite subir aplicaciones para ejecutarlas, proveer servicios y básicamente todo lo necesario desde el punto de vista del usuario. Otra manera de interactuar con CloudFoundry es a través de un entorno de desarrollo. Una vez terminamos el desarrollo de un módulo o una aplicación completa, queremos probar las modificaciones que hemos realizado en nuestra aplicación, y para ello, debemos desplegarla en un entorno de ejecución. CloudFoundry dispone de un plugin para Eclipse para poder realizar las tareas más comunes a la hora de desplegar una aplicación, sin tener que utilizar la línea de comandos.

Debemos destacar el IDE JazzHub. Dentro del entorno de desarrollo web para aplicaciones cloud de JazzHub, ya existe una integración natural con Bluemix, de forma que cuando decidamos que queremos desplegar nuestra aplicación, sólo haciendo click en la funcionalidad *deploy* y definiendo nuestras credenciales de acceso a Bluemix, procederá a desplegar nuestra aplicación. Una de sus grandes ventajas es su editor web basado en el editor Eclipse-Orion, que nos permite disponer de un entorno de gestión y desarrollo de aplicaciones sin tener ningún cliente o IDE local.

Al margen de saber cómo se gestiona una instancia de CloudFoundry o Bluemix, debemos conocer algunos términos importantes:

- Buildpack: Un buildpack es un componente que permite obtener los artefactos de una aplicación construido en un lenguaje de programación y sabe qué tiene que hacer para desplegar un entorno de ejecución e instalar y lanzar la aplicación entre otras cosas.

  Hay varios build packs disponibles y soportados por Bluemix, aunque la lista no deja de crecer:

  - Liberty for Java
  - NodeJS

- Noop
- Java
- Ruby
- GO
- Python
- PHP
- ASP.net
- XPages

Eso no quiere decir que no podemos utilizar otros buildpacks disponibles para CloudFoundry, simplemente que no están soportados oficialmente por IBM y podríamos encontrar algún problema o alguna incompatibilidad. El buildpack se utiliza a la hora de desplegar una aplicación, se especifica como parámetro en el cliente de la línea de comandos.

- Service: Un service es una funcionalidad que está disponible para nuestra aplicación. Hay muchos servicios ya disponibles en Bluemix. A continuación enumeraremos algunos ejemplos de servicios clasificados por su categoría:

  - Bases de datos SQL: SQLDB (Basado en DB2), mysql, postgresql
  - Bases de datos no SQL: Cloudant, JSONDB, mongodb
  - Sistemas de colas: ElasticMQ, rabbitmq
  - Servicios de BigInsights: BLUAcceleration
  - Servicios específicos para desarrollo de aplicaciones para moviles: MAM, CloudCode, MobileData
  - Servicios de caché de sesión: DataCache, SessionCache, redis

Tal como muestra la figura 2.5, la arquitectura de CloudFoundry está dotada de varias capas. A continuación vamos a detallar lo que hay por debajo de la capa de servicios de CloudFoundry explicando cada uno de sus componentes.

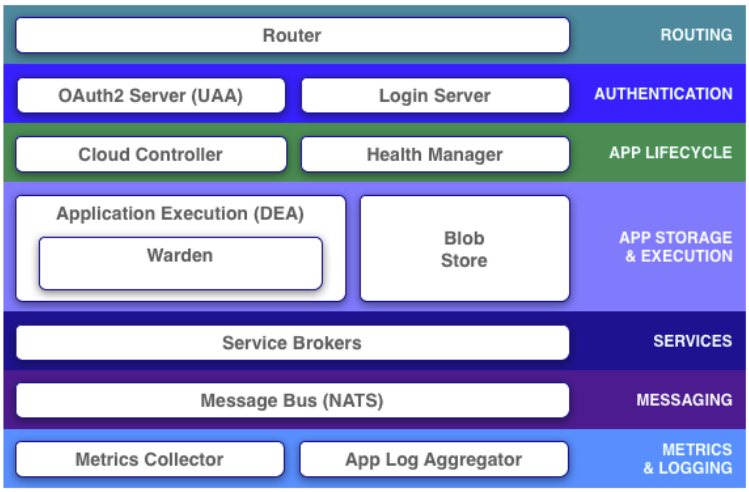

Figura 2.5: Arquitectura de CloudFoundry

Los componentes de la arquitectura son los siguientes:

- Capa de enrutamiento: El router es un componentes que se encarga de redirigir el tráfico hacia el Cloud Controller o bien la aplicación que le corresponda. Funciona como un servidor *http* que redirige peticiones. Cuando se inicializa una instancia completa de CloudFoundry, las peticiones sólo viajarán al Cloud Controller, pero cuando se despliegue la primera petición, sabrá que dependiendo de la *URL* que le entre, tendrá que redirigirlo a esta.
- Capa de autenticación: Se encarga, tal y como indica su nombre, de la autenticación. En esta capa tenemos dos componentes.
  - El servidor OAuth2: Que se encarga de llevar un control de las cuenta de usuario y de su autenticación.
  - El servidor de Login: Que nos permitirá personalizar los flujos de acceso (login) y otros como registro, cambio de password, etc.
- Capa de ciclo de vida de la aplicación: La principal finalidad de esta capa es controlar el ciclo de vida de las aplicaciones, su estado, su salud, y gestionar otros datos relativos a los despliegues, como las organizaciones, los espacios, servicios, instancias de los servicios y roles de los usuarios. Los dos componentes diferenciados son:
  - Cloud Controller: Gestiona el ciclo de vida de una aplicación, desde el momento que la importamos se almacenan los metadatos relativos a ella, así como los detalles de las instancias, servicios que consumen, etc. Y gestiona otro tipo de información como los entornos que tengamos definidos, estructuras organizativas, etc.
  - Gestor de Salud: Se comporta como un autómata que monitoriza el estado de las aplicaciones desplegadas, en qué estado debería estar, y si no coinciden, se toman medidas.
- Capa de almacenamiento y ejecución de aplicaciones: Esta capa es la que tiene la responsabilidad de almacenar los artefactos de cada aplicación y de ejecutarlas. Aquí es donde realmente se trabaja para hacer funcionar la aplicación. Los dos componentes con entidad son:
  - Droplet Execution Agent: es un controlador que gestiona los contenedores Warden. Un contenedor Warden es un componente que se encarga de crear entornos efímeros, aislados y con recursos limitados.
  - Almacén Blob: es un contenedor para almacenar los objetos binarios de las aplicaciones, los droplets (aplicaciones ya transformadas y preparadas para ser desplegadas) y los buildpacks.
- Capa de Servicios: en esta capa se definen los diferentes brokers de servicios y proveen instancias a las aplicaciones que los necesitan o los tienen asociados. Tenemos que pensar que un Service Broker actúa como un broker más que como una puerta de enlace. El broker por lo general proporcionará un acceso y se desentiende de su explotación, es decir, la aplicaciones se conectará con estos datos de acceso, mientras que una puerta de enlace nos proporciona un acceso y nos obliga a pasar por ella cada vez que tenemos que acceder al servicio.
- Capa de mensajes: se encarga de intercambiar mensajes entre los diferentes componentes de CloudFoundry a nivel interno.
- Capa de utilidades: un componentes que nos da métricas de los componentes (ej: RAM consumida) y *Logs* relativos a la aplicaciones respectivamente.

2.3.3. Node-RED

Node-RED es una herramienta que permite interconectar dispositivos de hardware distintos, API's y servicios en línea a través de una interfaz sencilla mediante soluciones software.

Node-RED ofrece un editor visual de flujos basado en el navegador que permite de manera muy sencilla enlazar los flujos de datos utilizando una gran cantidad de nodos que permiten manejar dichos flujos. Los flujos se pueden desplegar en tiempo de ejecución, o mediante una peticiones, o cuando se realiza un determinado evento.

Gracias a Eclipse Orion, se pueden desarrollar funciones de JavaScript para ampliar las funcionalidades de los nodos y los flujos. Dispone de una biblioteca para almacenar las funciones que se desarrollan en estos nodos para reutilizarlos cuando sea necesario.

En la figura 2.6, podemos observar los nodos básicos que vienen incluidos en la aplicación. Debemos destacar que se pueden implementar nuevos nodos en función de nuestras necesidades, que en nuestro caso, han sido los nodos de geolocalización.

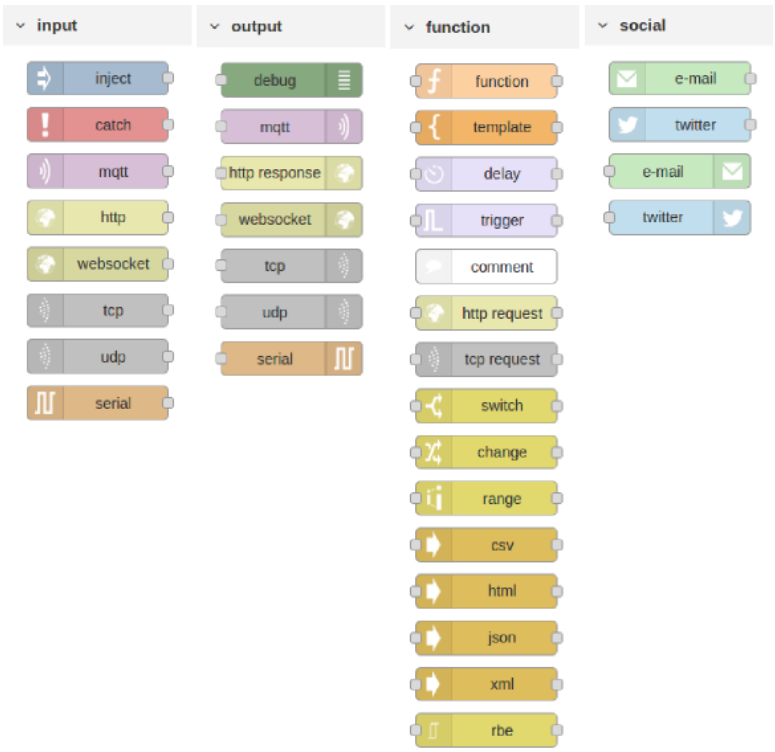

Figura 2.6: Nodos básicos de Node-RED

2.3.4. Geospatial Analytics

Este servicio nos permite supervisar los dispositivos que estén en un ámbito geográfico de interés desde la plataforma Bluemix. El servicio Gespatial Analytics se puede utilizar en aplicaciones que dan soporte a diferentes casos de uso, de forma que notifica a usuarios que están cerca de una ubicación específica o que situaciones específicas estén cerca del usuario.

El servicio utiliza el lenguaje de programación *Javascript* en el entorno de tiempo de ejecución *Node.js* debido a su bajo tiempo de ejecución. El paso de mensajes se realiza a través del protocolo de mensajes MQTT para facilitar datos de dispositivos geospaciales y recibir notificaciones de eventos para dispositivos. MQTT es un protocolo de mensajería ligero idóneo para aplicaciones

móviles. Cuando Geospatial Analytics detecta que un dispositivo entra en cualquiera de las regiones geográficas que va a supervisar o sale de ella, envía un evento a MQTT.

Se deben cumplir algunos requisitos específicos para que el servicio funcione correctamente:

- El intermediario de mensajes MQTT debe publicar mensajes de dispositivo en formato JSON en uno o varios tópicos MQTT.
- Los mensajes de dispositivo pueden contener varios atributos, sin embargo, es imprescindible que contenga un atributo que identifique el dispositivo, un atributo que especifique la latitud, y un atributo que especifique la longitud de la ubicación actual del dispositivo.
- Un mensaje MQTT puede contener un objeto JSON con información sobre un único dispositivo o una matriz de objetos JSON con información sobre un conjunto de dispositivos.

La aplicación controla el servicio geoespacial con una API REST. Las siguientes opciones están disponibles mediante llamadas API REST:

- Configurar e iniciar el servicio
- Añadir y eliminar regiones geográficas que se desea supervisar
- Recuperar información sobre el estado del servicio y sobre el conjunto de regiones definidas actualmente
- Detener el servicio

2.3.5. IBM Push Notifications

El servicio IBM Push Notifications mide el estado, el comportamiento y el contexto de las aplicaciones móviles, los usuarios móviles y los dispositivos móviles proporcionando servicios de supervisión para aplicaciones móviles que acceden a los recursos de nube alojados en Bluemix. Los registros de lado del cliente y los datos de uso se recopilan automáticamente y se envían al servicio de Mobile Analytics a petición. Los desarrolladores y administradores pueden utilizar el panel de instrumentos del servicio de Mobile Analytics para ver datos recopilados por el SDK del cliente.

El servicio de Notificaciones push proporciona una plataforma unificada para enviar y gestionar notificaciones push móviles que están pensadas para plataformas iOS y Android. Este servicio gestiona la correlación de los usuarios de la aplicación a sus dispositivos, plataforma de dispositivo, y maneja el envío de notificaciones push a los mismos. Con este servicio, puede enviar difusiones, unicasts (en función del ID de dispositivo) y también notificaciones push (o temas) de etiquetas a los usuarios de la aplicación móvil. También puede utilizar las API REST y SDK para desarrollar más las aplicaciones de cliente.

2.3.6. Cloudant NO-SQL Database

IBM Cloudant NoSQL DB for Bluemix es una base de datos NoSQL como servicio (DBaaS). Está creada desde la base para escalar globalmente, ejecutarse sin parar y manejar una gran variedad de tipos de datos como JSON, texto completo y geoespacial. Cloudant NoSQL DB es un almacén de datos operativos optimizado para manejar lecturas y escrituras simultáneas y proporcionar alta disponibilidad y durabilidad de datos.

2.4. Comparativa entre PaaS

En esta sección realizaremos una comparación entre las principales PaaS[17], considerando aspectos como:

- Lenguajes de programación: la selección del lenguaje o lenguajes de programación puede influenciar directamente en la selección de la PaaS que necesitamos. Los lenguajes influyen directamente en el paradigma de programación, en las herramientas y en los componentes que vamos a desplegar.
- Tecnología utilizada en el servidor: la tecnología utilizada en el servidor tiene una gran influencia en cómo diseñamos nuestra aplicación. Las más populares son .NET, PHP y Java y aparecen en la mayoría de los proveedores. Si nuestro desarrollo está centrado en .NET, deberíamos elegir una PaaS especializada en este entorno, sin embargo, si nuestro desarrollo contiene varios lenguajes de programación, deberíamos centrarnos en un entorno políglota.
- Almacenamiento de datos: debemos considerar las necesidades de almacenamiento de nuestra aplicación en cuanto a latencia, formato de almacenamiento, etc.
- Soporte e integración con herramientas y aplicaciones: es importante considerar si podemos integrar la PaaS con herramientas de desarrollo como Eclipse o Visual Studio, ya que una buena integración reduciría la sobrecarga y el tiempo de despliegue por parte de los desarrolladores, además de reducir los posibles errores durante el despliegue. También debemos considerar si nuestra aplicación puede integrarse fácilmente con otras aplicaciones cloud.
- Costes y presupuestos: es el aspecto menos técnico pero no por ello el menos importante. Es razonable que los costes de una PaaS sean mayores que los costes de una IaaS ya que las PaaS nos ahorran esfuerzos en cuanto a entorno, rendimiento y escalabilidad, características que debemos gestionar manualmente si tenemos una IaaS.

2.4.1. AWS Elastic Beanstalk

Amazon Web Services (AWS), lanzada oficialmente en 2006, es una colección de servicios de computación en la nube que en conjunto forman una plataforma de computación en la nube, ofrecidas a través de internet por Amazon. Es de las ofertas internacionales más importantes de la computación en la nube.

AWS Elastic Beanstalk es la plataforma como servicio de Amazon Web Services. Los lenguajes de programación admitidos son PHP, Java, Python, Ruby, Node.js, .NET, Go. Además permite la instalación de dockers que puedan ejecutarse en un servidor de aplicaciones con una base de datos.Elastic Beanstalk utiliza Auto Scaling y Elastic Load Balancing para administrar de manera sencilla cantidades de tráfico muy variables. Puede comenzar con poco y escalar la capacidad. Se requiere una mayor gestión debido a que está integrado con AWS, con lo cual aumenta la sobrecarga en comparación con otras PaaS, ya que tenemos que encargarnos de algunas partes de la infraestructura. Los costes son complejos ya que se basan en muchas variables como las instancias, el almacenamiento, los servicios de aplicación o sobrecarga de datos. Tenemos la ventaja de que nos ofrece una calculadora que permite verificar con detalle el coste de nuestra aplicación. No se aplican cargos adicionales por utilizar AWS Elastic Beanstalk. Solo tiene que pagar por los recursos de AWS creados para almacenar y ejecutar la aplicación. Solo pagará por lo que consuma y a medida que lo utilice: no se requieren pagos mínimos ni compromisos iniciales.

2.4.2. Microsoft Azure

Microsoft Azure es una plataforma ofrecida como servicio y alojada en los Data Centers de Microsoft. Anunciada en el Professional Developers Conference de 2008 en su versión beta, pasó a ser producto comercial el 1 de enero del 2010. Microsoft Azure es una plataforma general que tiene diferentes servicios para aplicaciones, desde servicios que alojan aplicaciones en alguno de los centros de procesamiento de datos de Microsoft para que se ejecute sobre su infraestructura hasta servicios de comunicación segura.

Azure ha ensamblado su IaaS y PaaS en una sola plataforma, con lo cual permite a los desarrolladores tener un mayor control de la aplicación y su arquitectura. Como limitaciones, tenemos un portal de gestión y administración minimalista. Sus costes se basan en el tamaño de las instancias que se están ejecutando. Los precios pueden variar entre 0.02$/hora (768MB de RAM y 1 núcleo virtual compartido) 0.64$/hora (14GB de RAM y 8 núcleos virtuales). El servicio es gratuito durante 30 días con un bonus de 200$ para consumir en servicios, solamente para nuevos usuarios.

2.4.3. Google App Engine

Google App Engine es otro de los servicios que conforman la familia de Google Cloud Platform. Este servicio es una PaaS que nos permite publicar aplicaciones web en línea sin necesidad de preocuparnos por la parte de la infraestructura y con un enfoque en el desarrollo de la aplicación.

Google App Engine utiliza un modelo sandbox que aísla los procesos reduciendo así el riesgo de que un proceso defectuosos en un servidor físico pueda interferir negativamente en otros procesos del servidor. Como aspectos negativos, nos encontramos el hecho de que los lenguajes de programación están limitados a Java, Python, Go y PHP. Los costes son en base al uso. Based on usage. 0.08$/hora por demanda de la instancia o 0.05$/hora por una instancia reservada. El almacenamiento de datos oscila entre 0.18$/GB mensuales. El coste de ancho de banda es de 0.12$/GB. Se puede aplicar costes adicionales por otros servicios. Las primeras 28 horas de cada instancia, 1GB de almacenamiento de tráfico entrante o saliente por aplicación son gratuitos y se renuevan por día. Microsoft ofrece una opción para aquellos alumnos que quieran realizar su proyecto de fin de carrera con Azure, y proporciona un año de servicio gratuito.

3

Sistema, diseño y desarrollo

3.1. Introducción

En esta sección, detallaremos las directrices que hemos seguido para desarrollar cada módulo de la aplicación que utiliza lo que hemos aprendido durante la fase de investigación sobre Internet of Things, cloud computing y aplicaciones móviles.

En resumen nuestra aplicación es un detector de incidencias en función de la geolocalización de un usuario, el cual define un radio kilométrico de forma que solamente las incidencias que se encuentren dentro de la zona geográfica establecida sean mostradas al usuario.

Como podemos ver en la figura 3.1, el flujo general de la aplicación es básico, sin embargo los flujos internos que contiene la plataforma cloud son mucho más complejos y los veremos explicados en detalle en su correspondiente sección.

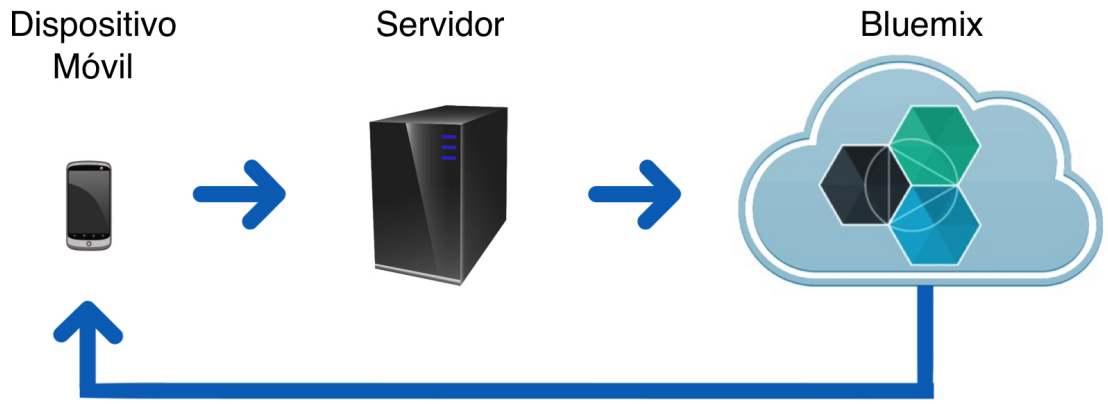

Figura 3.1: Flujo general de datos de la aplicación

El flujo se inicia al realizarse una petición desde el dispositivo móvil en la que se incluye la geolocalización del dispositivo. Esta petición va dirigida a un servidor construido en una

RaspberryPi, dispositivo que consume muy pocos recursos, su pequeño tamaño y su capacidad de cálculo. El servidor construye un mensaje MQTT, el cual es enviado a un broker mediante un tópico. Dicho mensaje es enviado a la plataforma Bluemix, la cual inicia un flujo en el que solicita datos a una API de la dirección general de tráfico, y en función de los datos de latitud y longitud, retorna al usuario aquellas incidencias de interés para el usuario.

3.2. Aplicación IOT Android

El concepto de Internet of Things viene determinado por todos los dispositivos capaz de emitir o recibir datos a través de internet u otros tipos de conexiones, tal como hemos visto en la sección 2.1. Partiendo de este punto, hemos determinado que un dispositivo móvil puede ser un buen ejemplo para esta aplicación ya que contiene varios sensores, y uno de ellos que es el que nos interesa, el sensor de geolocalización espacial.

En primer lugar, debemos obtener los datos de geolocalización del dispositivo móvil. Los datos se obtienen a partir de la clase LocationManager.

Posteriormente debemos construir el sistemas de envío de mensajes MQTT. Para ello debemos inicializar el servicio, en segundo lugar, debemos conectarnos a un broker para realizar el envío de mensajes. El broker funciona exactamente con el sistema de hashtags de Twitter, en el que nos suscribimos a un usuario o hashtag y cada notificación que contenga esta etiqueta la recibiremos mediante una notificación. De la misma forma, nos suscribimos a un broker en el que estaremos escuchando cada notificación que llegue a la etiqueta definida.

La primera ventana de la aplicación se muestra en la figura 3.2, y tiene una sencilla interfaz que envía la localización geoespacial al servicio desarrollado en la plataforma Cloud.

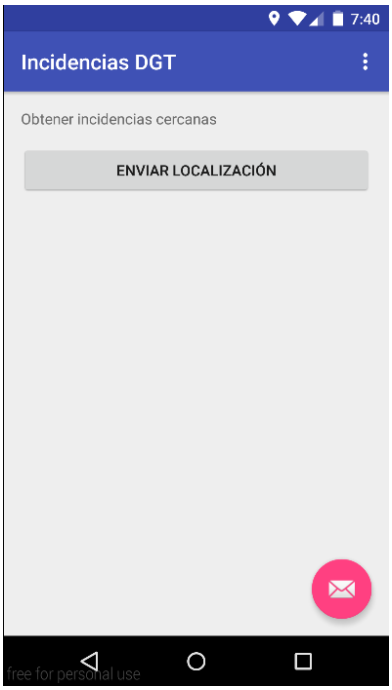

Figura 3.2: Primera vista de la aplicación móvil

Una vez enviado los datos de geolocalización, inicializaremos una nueva actividad que utiliza un LinearLayout con una lista que contiene vistas con las incidencias de tráfico que recibiremos. Si seleccionamos la incidencia, podremos ver los detalles de la misma en nuestro dispositivo móvil tal como podemos visualizar en la figura 3.3.

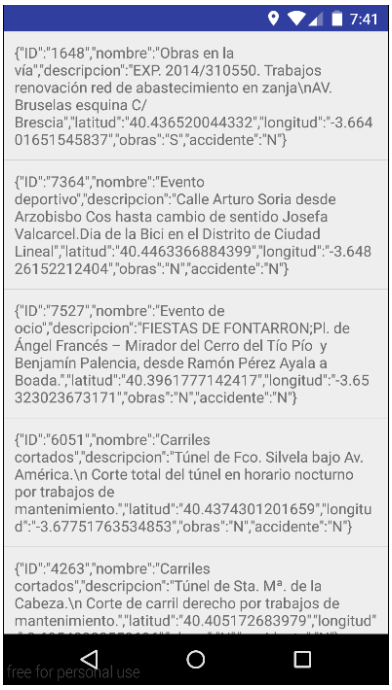

Figura 3.3: Segunda ventana de la aplicación móvil

Es interesante que la aplicación Android es el punto inicial del flujo de datos y a la vez, es el punto final de la aplicación. Sigue un modelo consumidor/proveedor en el que un cliente solicita una información y mediante *cloud computing* generamos la información para el usuario.

3.3. Creación de la aplicación base

En esta sección explicaremos paso a paso cómo hemos implementado la aplicación base. Aunque los servicios proporcionados por Bluemix se pueden utilizar con cualquiera de los lenguajes disponibles a través de la API REST que proporcionan, en este caso, para simplificar el desarrollo utilizaremos Node-RED.

Node-RED proporciona un editor de flujos basado en el navegador web que facilita la interconexión de dispositivos, API's y servicios en línea mediante el uso de la amplia gama de los nodos de la paleta. Los flujos pueden ser desplegados en el *runtime* Node.js de manera instantánea. Esta aplicación de arranque proporciona una versión del Node-RED personalizado para funcionar en IBM Bluemix.

Tras haberse registrado en Bluemix, procedemos a crear una nueva aplicación utilizando un contenedor modelo el cual funciona como una plantilla a partir de la cual desarrollaremos nuestra aplicación. Nos dirigimos al catálogo de servicios de Bluemix y seleccionamos la aplicación Node-RED Starter. Los contenedores modelo se pueden visualizar en la figura 3.4.

Cuando seleccionamos el boilerplate de "Node-Red Starter" debemos rellenar el nombre de la aplicación y automáticamente se generará el nombre del host de la aplicación (dónde se va a desplegar la aplicación). Podemos visualizar este proceso en la figura 3.5.

Observamos que se genera el SDK para Node.js y una base de datos de Cloudant NoSQL DB. Aunque este servicio no lo vamos a necesitar por ahora, es imprescindible para el funcionamiento de la aplicación, con lo cual, es recomendable no borrarlo.

Al pulsar el botón "create", pasaremos a la etapa de despliegue de nuestra aplicación. Podemos

Figura 3.4: Contenedores modelo de Bluemix

Figura 3.5: Configuración módulo Node-RED

salir de esta ventana e ir a nuestro "dashboard" (panel de control) en el que podemos ver todas nuestras aplicaciones y servicios asociados.

Una vez finaliza el proceso de creación, tenemos un entorno de ejecución listo para trabajar, con la base de datos creada y con la URL dada de alta y accesible desde Internet.

3.4. Servidor de Mensajes

Los mensajes con la información que contiene los datos sobre la localización espacial del dispositivo, en una primera versión, eran procesados por un servidor construido sobre una *RaspberryPi* debido a su bajo consumo de energía y a su velocidad de procesamiento de datos. En primera instancia, todo funcionaba correctamente, sin embargo, se estaba generando un cuello de botella en este punto debido a que ante un posible incremento en el número de usuarios de la aplicación, supondría un aumento considerable de las peticiones al servidor.

A primera vista podríamos pensar que dicha situación supone una ventaja ya que estamos

eliminando la carga de procesamiento de la plataforma como servicio, con lo cual, significa un ahorro de costes dado a que no estamos utilizando dicho servicio en la nube. Sin embargo, esto sería factible ante una carga muy baja de peticiones y usuarios. A medida que la aplicación incrementase el número de peticiones, nos encontraríamos con el cuello de botella mencionado anteriormente, y deberíamos escalar manualmente nuestro sistema para poder soportar tal cantidad de peticiones.

La alternativa más sencilla y más coherente, fue utilizar la nube como plataforma para administrar las peticiones. De esta forma todas nuestras aplicaciones están gestionadas por la plataforma. En esta sección vemos una de las grandes ventajas de las plataformas *cloud*, y es la escalabilidad. *Bluemix* nos garantiza el normal funcionamiento de la aplicación independientemente del número de usuarios o de la cantidad de datos a ser procesados.

Pongamos un ejemplo, si el tiempo de respuesta para cada petición es de cinco milisegundos, al haber un millón de usuarios realizando peticiones simultáneas, el tiempo de respuesta no sufriría ninguna alteración, ya que la plataforma *cloud* escala automáticamente para mantener el rendimiento deseado.

El servidor se ha implementado utilizando el lenguaje de programación *php* utilizando los servicios proporcionados por *Bluemix*. El panel de control del servidor tiene la siguiente interfaz en la que podemos modificar sus parámetros manualmente si deseamos, tal como podemos observar en la figura 3.6.

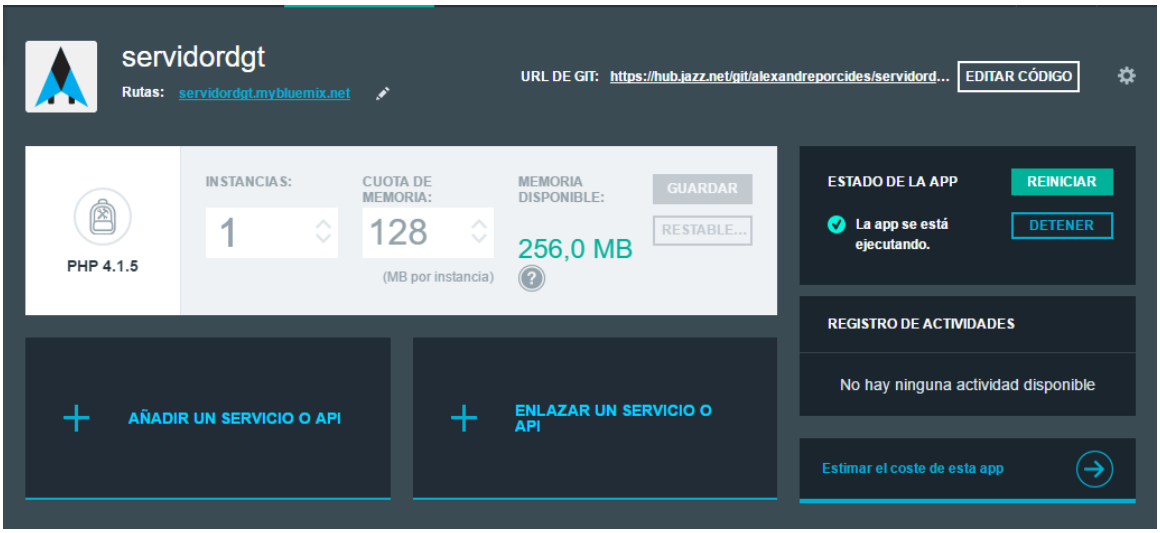

Figura 3.6: Panel de control del servidor

El servidor recibe una petición desde el móvil la cual incluye su localización espacial, y el servidor construye un mensaje JSON, que posteriormente será enviado mediante el protocolo MQTT al entorno de *Internet of Things* Node-RED, el cual iniciará el flujo de funcionamiento de la aplicación. A partir de este punto, la información será utilizada por el gestor de flujos y los servicios de geolocalización de la plataforma para devolver información relevante al usuario final en su dispositivo móvil.

3.5. Servicio de Geolocalización

Realizaremos el servicio de geolocalización para monitorizar los objetos que entren o salgan de una determinada zona geográfica. Para ello, vamos a añadir este servicio desde el menú de nuestra aplicación:

Figura 3.7: Servicio de geolocalización

Al seleccionar este servicio, debemos proporcionarle un nombre y asegurarnos de que el servicio estará en asociado a nuestra aplicación. Podemos asegurarnos de ello observando el "text box" de "app" y "space".Al finalizar se nos solicitará que volvamos a desplegar la aplicación.

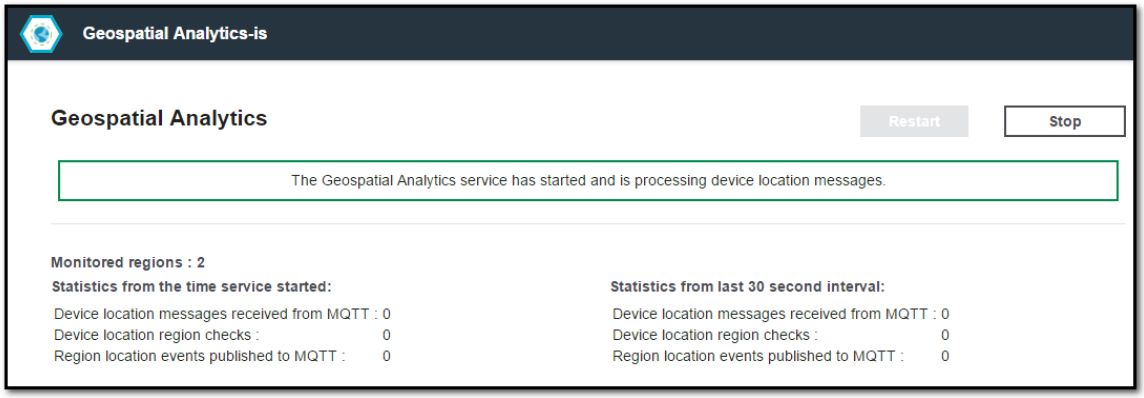

Figura 3.8: Panel de visualización de datos Geospatial Analytics

En el panel de visualización de datos geoespaciales (Figura 3.8), podemos visualizar las regiones que están siendo monitorizadas por nuestra aplicación y el número dotal de mensajes recibidos, enviados y el número de verificaciones realizadas por una región determinada. A cada 30 segundos se está actualizando esta información.

Para utilizar este servicio en plataforma de control de flujos IoT Node-RED, debemos implementar los nodos de análisis geoespaciales, ya que estos no vienen por defecto en la plataforma Node-RED. Un nodo de Node-RED consiste en un conjunto de dos ficheros, uno con extensión *.js* en el que va la lógica del nodo programada en Javascript, y el otro fichero es un archivo con extensión *.html* en el que se configuran los parámetros de entrada y la parte frontend del nodo. Para facilitar el desarrollo de los nodos que en principio no es muy intuitivo, Bluemix nos proporciona una plantilla desde la que partimos y desarrollamos los nodos.

Una vez implementado los nodos, debemos insertar los nodos en el proyecto, y se realiza a través del IDE online JazzHub. Si insertamos correctamente los nodos y no se produce ningún tipo de error, nuestros nodos encargados de monitorizar y configurar la geolocalización estarán en la plataforma Node-RED, dispobibles en el panel de nodos.

Figura 3.9: Botón de edición Node-RED

3.6. Control de Flujos con Node-RED

En el Overview de la aplicación tenemos un enlace que nos llevará al entorno de desarrollo Node-RED que forma parte de la propia aplicación. En esa interfaz vemos la url que se ha generado para nuestra aplicación y en la cual se va a poder acceder al servicio.

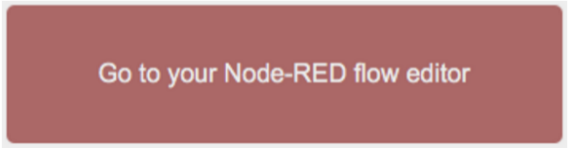

Figura 3.10: Nodos personalizados de geolocalización

Tal como hemos visto en la sección 2.3.2, Node-RED funciona con un sistema de flujos a través del paso de mensajes con un valor muy bajo de sobrecarga en la red. El punto inicial del sistema de nodos es la información de la geolocalización proveniente del dispositivo móvil.

Cuando abrimos la aplicación, observamos que a la izquierda tenemos la paleta de componentes con los que construiremos la aplicación. A la derecha información sobre los componentes utilizados e información de depuración(logs) y en el centro, el espacio de trabajo.

Cada vez que queramos probar un cambio en el código es necesario pulsar el botón *Deploy* en la parte superior derecha. Sabremos que hay cambios sin guardar porque el botón esta activo y los componentes modificados muestran un pequeño círculo de color azul. Un pequeño triángulo de color rojo indica un error de configuración en el componente. Si no pulsamos *Deploy* no se guarda el proyecto. En caso de caída del navegador, perderíamos el trabajo. Es recomendable ir desplegando periódicamente aunque no vayamos a probar y haya errores en el proyecto. (componentes sin configurar).

Las aplicaciones en Node-RED se modelan como flujos entre componentes. La información se comparte entre esos componentes mediante un objeto llamado *msg*. En este objeto podemos guardar los datos sin necesidad de definirlos previamente(estilo Javascript). La mayoría de los componentes utilizan la propiedad *msg.payload* de este objeto para recuperar los datos de entrada y guardar los datos de salida.

En esta sección hemos desarrollado los flujos necesarios para el correcto funcionamiento de la aplicación. A continuación detallaremos cada uno de estos flujos de datos.

3.6.1. Flujo de obtención y procesado de datos

El primer flujo obtiene datos en tiempo real relativos a incidencias en la ciudad de Madrid a partir de los datos proporcionados por la Dirección General de Tráfico.

El flujo puede iniciarse a través de una actividad por parte del dispositivo móvil o manualmente a través del editor web de Node-RED. A continuación se realiza una petición de los datos al servidor de la DGT. Los datos proporcionados por la DGT están en formato xml, y lo trasformamos a Json mediante una librería instalada a través de npm, un gestor de paquetes de Javascript. Una vez tenemos los datos en Json, excluimos todos los metadatos que no son relevantes para el análisis tal como la versión del documento u otras características.

Lo que tenemos ahora es un array de incidencias, y lo que haremos es iterar una a una para enviarlas en orden al servicio de geolocalización. Una vez terminado el envío, este módulo entra en modo de espera, esperando otras peticiones al servicio. Guardamos solamente los campos que

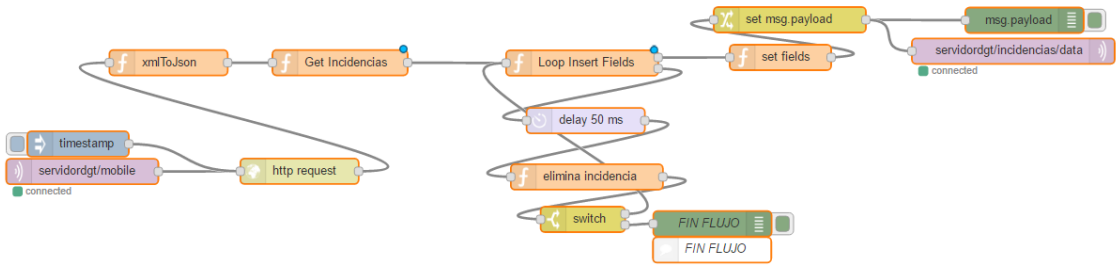

Figura 3.11: Flujo general de datos Node-RED

son interesantes al usuario final, y mediante el protocolo MQTT, publicamos los mensajes en un broker. Los elementos o aplicaciones suscritas a este broker recibirán los mensajes.

Los nodos de color naranja, señalados con una *f* son nodos de funciones. Estos nodos reciben un mensaje JSON como parámetro de entrada, y nosotros como desarrolladores procesamos la información del mensaje y generamos una salida. Todo ello se realiza mediante el lenguaje Javascript. A continuación podemos observar un ejemplo de como codificar la funcionalidad de este nodo:

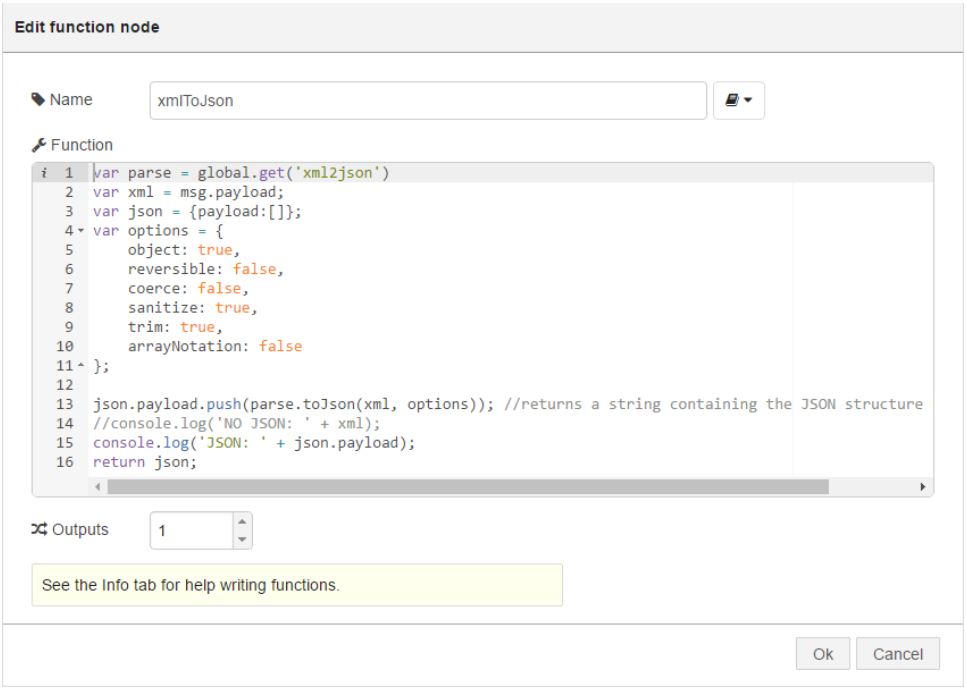

Figura 3.12: Conversion de un mensaje xml a Json en Javascript

3.6.2. Flujo de análisis de geolocalización

El flujo de datos geolocalizables son nodos que configuran el entorno del servicio de la geolocalización. *Start Service* se inicia automáticamente cuando la aplicación se inicializa, y se encarga de suscribirse a uno o varios brokers de los cuales recibirán mensajes MQTT para pasar a su posterior procesado. Se definen aquellos campos relacionados con la geolocalización, es decir, los nombres de las variables que almacenan la latitud y la longitud de la incidencia que se recibirá a través del protocolo MQTT. Tras el análisis geoespacial, se enviará un mensaje con la incidencia que cumpla los requisitos definidos a un broker, el cual tendrá sus correspondientes suscriptores.

Stop Service pausará el servicio indeterminadamente hasta que se vuelva a inicializar. Otro de los nodos importantes en la aplicación es el nodo indicado con "*Add lat, lon*", que define una región geográfica como punto de análisis, que en nuestra aplicación procederá de los datos gps del dispositivo móvil que esté utilizando la aplicación. El servicio establece un radio de acción en el que descarta aquellas incidencias que estén fuera del rango establecido por el usuario. *Status* muestra el estado del servicio de geolocalización, y *Remove region* elimina la región añadida para el análisis.

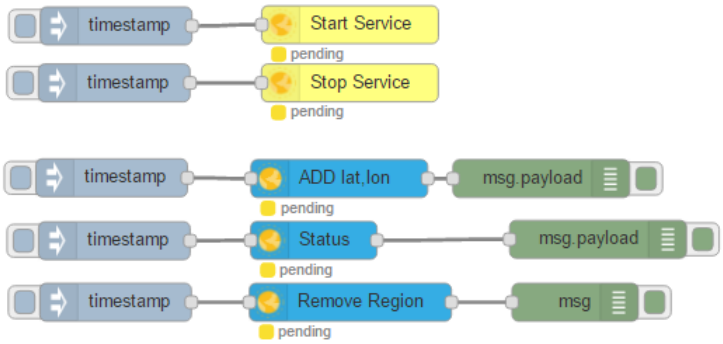

Figura 3.13: Flujo de análisis de geolocalización

3.6.3. Flujo de recepción de datos

El flujo de recepción de datos está suscrito al broker que publicará los mensajes filtrados por el servicio de geolocalización. Mediante un convertidor de array json a objeto json, facilitaremos el acceso a cada uno de los valores del mensaje, y con ello, generaremos un mensaje comprensible al usuario final. La aplicación tiene dos puntos de salida, el servicio de IBM Push y un broker MQTT.

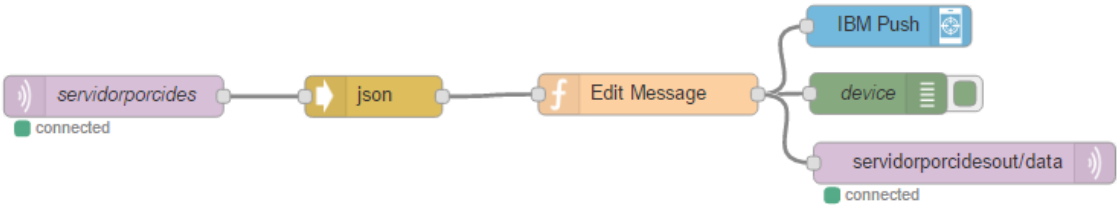

Figura 3.14: Flujo de recepción de datos

El servicio de IBM Push mostrará notificaciones push en los dispositivos que tengan instalado el servicio y estén suscritos a la aplicación. El broker MQTT final de la aplicación, es el broker al que está suscrito nuestra aplicación móvil que hemos desarrollado en la sección 3.2.

4

Experimentos Realizados y Resultados

4.1. Escenario de pruebas

El escenario de prueba en el que se ha probado la aplicación, ha sido un sistema con 1 nodo y 512MB de memoria. Una de las ventajas de Bluemix es la autoescalabilidad. El servicio de autoescalabilidad es gratuito y gestiona el sistema de manera automática de forma que ante la necesidad del sistema de más memoria o instancias de la aplicación, el servicio siga funcionando correctamente.

Figura 4.1: Configuración inicial de recursos

Los recursos que disponemos en Bluemix en la versión gratuita, son 2GB de memoria y un número indeterminado de instancias que pueden repartir los 2GB mencionados anteriormente. Es decir, podemos tener una aplicación con 4 instancias de 512MB.

Como el sistema está desarrollado de manera modular, las pruebas de resultados se han realizando de manera independiente. Los módulos están desarrollados en el entorno Node-RED, el cual ofrece dos opciones para verificar que los nodos están haciendo sus operaciones correspondientes. La primera opción y más sencilla visualmente es utilizar un nodo debug. El nodo debug permite verificar la información que está pasando por un punto del sistema en un momento determinado. Es de la siguiente manera:

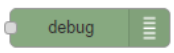

Figura 4.2: Nodo debug

Para visualizar los datos y verificar que el proceso se está ejecutando correctamente a través de los nodos debug, debemos acceder a la sección de debug mediante su correspondiente pestaña en la plataforma Node-RED. Veremos la información contenida, la fecha en la que se ha publicado la información y el tipo de objeto que está pasando por el nodo en este determinado momento.

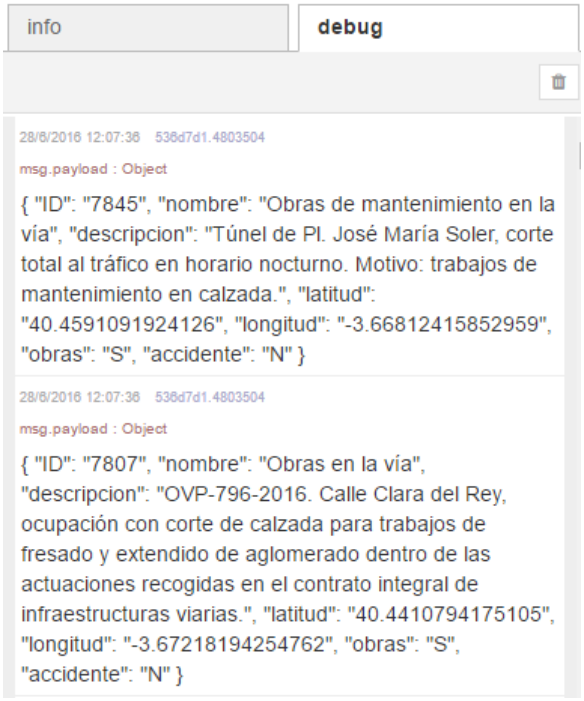

Figura 4.3: Contenido de un nodo debug en un determinado instante

La segunda opción para depurar el sistema, es escribir en el archivo *log*. En ese fichero se registran todas las actividades del sistema, con lo cual es recomendable utilizar etiquetas que identifiquen la sección que estamos depurando. Para escribir en el log del sistema, debemos utilizar el comando *console.log()* en la sección de código. Cualquiera de las opciones de depuración y comprobación de errores son válidas y durante este desarrollo se han utilizado las dos opciones.

Cuando trabajamos con una PaaS, y utilizamos los servicios que nos proporcionan, dependemos del buen funcionamiento de estos. Durante el desarrollo de la aplicación, nos hemos encontrado que uno de los servicios que estábamos utilizando para enviar notificaciones al dispositivos móvil iba a dejar de existir en la plataforma. El aspecto negativo es el hecho de que nos han advertido con anterioridad que el servicio iba a dejar de existir y nos han planteado alternativas posibles para solventar la desaparición del servicio afectado, y de esta manera hemos procedido.

4.2. Experimentos del sistema completo

El experimento del sistema completa consiste en probar la aplicación en el dispositivo móvil, la cual activará todo el sistema desarrollado y mostrará por la pantalla del dispositivo móvil los resultados obtenidos tras el proceso de obtención y filtrado de información. Al estar trabajando con una plataforma que se ha ido actualizando y sufriendo cambios durante la fase de desarrollo, nos hemos encontrado con situaciones que la propia PaaS ha solventado de manera satisfactoria.

Una de las situaciones que han sucedido y nos han servido de prueba del sistema es el hecho de que si la PaaS Bluemix por alguna razón sufre una caída del sistema, ¿qué ocurre con nuestras

aplicaciones? En dos etapas del desarrollo nos hemos encontrado con esta situación en la que Bluemix estaba inaccesible por motivos de mantenimiento o por un posible error interno. Nos ha llamado la atención que a pesar de que la plataforma estaba inaccesible, nuestra aplicación seguía funcionando con la última versión estable. Es una gran ventaja ya que en ese sentido no hay una dependencia con la plataforma. Sin embargo, durante los periodos de actualizaciones o mantenimientos de la PaaS, no hemos podido añadir modificaciones a nuestro sistema.

Supusimos el caso en el que nuestra aplicación necesitara pasar por una actualización importante, y tenemos que volver a desplegar en producción nuestra aplicación actualizada. Al tratarse de una PaaS, contamos con la ventaja de que el tiempo de configuración y despliegue en la nube ronda alrededor de pocos minutos. Es importante saber que la infraestructura y el entorno es proporcionado por el proveedor de servicios.

Las pruebas realizadas en la aplicación se han dado de forma satisfactoria, mostrando las incidencias correspondientes al usuario final. De igual manera en la aplicación secundaria que consistía en la rastreo de vuelos dentro de un área geográfica funciona de manera correcto. Con lo cual, es evidente que el servicio se puede extender a aplicaciones distintas, y seguirá funcionando correctamente.

5

Conclusiones y trabajo futuro

5.1. Conclusiones

El objetivo principal de este trabajo era desarrollar una aplicación sobre una plataforma cloud, analizando datos de IoT y servicios de geolocalización. Lo hemos desarrollado todo de manera modular utilizando aplicaciones de ejemplo en cada módulo de la aplicación. Hemos alcanzado este objetivo de manera satisfactoria.

Durante el desarrollo de este proyecto hemos adquirido conocimientos acerca de plataformas cloud, y más en profundidad a PaaS de IBM, Bluemix. Durante este periodo de investigación y desarrollo, hemos aprendido y comprobado las posibilidades y limitaciones que nos ofrece Bluemix como PaaS. A través de Bluemix se ha llegado a comprender el funcionamiento de los servicios de geolocalización de IBM, Geospatial Analytics, el cual ha sido uno de los puntos más importantes de la aplicación desarrollada. A su vez, Bluemix dispone de un módulo de Internet of Things que nos ha permitido interconectar y gestionar dispositivos y datos. Hemos adquirido conocimiento en cuanto a sistemas empotrados, como la RaspberryPi, en la cual hemos construido un servidor de peticiones para nuestra aplicación. Internet of Things está relacionado con el mundo de los sensores, y durante este desarrollo hemos utilizado los sensores gps que tienen los dispositivos móviles, y de esta forma, relacionarlos con la geolocalización.

Debemos destacar que la aplicación funciona de manera gratuita, y no nos supone ningún tipo de gasto. Sin embargo, al tratarse de una PaaS, si llegáramos a superar los límites de uso, peticiones, servicios o almacenamientos establecidos en la versión gratuita, estaríamos pagando a medida que la aplicación fuera creciendo. Esto nos lleva a una de las grandes ventajas de las PaaS, y es el hecho de que son autoescalables, proporcionado el rendimiento óptimo para cada usuario independientemente del número de usuarios del servicio.

A través de cualquier API o sensores que estén emitiendo información en tiempo real, es posible realizar una aplicación de análisis de datos, utilizando el módulo de internet of things de Bluemix. Nos proporciona muchas posibilidades de salidas de datos a través de sus nodos, por ejemplo, correo electrónico, Twitter, mensajes MQTT, envío de datos a un servidor TCP o UDP, envío de una petición http, mensajes a dispositivos móviles u otras alternativas. Node-RED además nos da la posibilidad de generar nuevos nodos en función de nuestras necesidades, tal como han sido los nodos de geolocalización espacial que hemos construido.

Como aportación personal, este trabajo de fin de carrera ha sido muy importante ya que he desarrollado conocimientos en lenguajes de programación con los que no tenía mucha experiencia, como es el lenguaje de Javascript. De igual manera, haber tenido una primera experiencia con PaaS me ha permitido conocer las últimas tecnologías en la que están apostando grandes empresas a nivel mundial tal como IBM, Google, Amazon o Microsoft.

5.2. Trabajo Futuro

La aplicación consiste en una primera versión estable que permite obtener datos de incidencias cercanas a un punto geográfico determinado delimitando los resultados a través de un radio geográfico. Este radio está definido como variable estática en la aplicación de la plataforma cloud. Una versión más avanzada sería la posibilidad del usuario de definir su radio de interés y poder filtrar la información recibida en función de las variables del mensaje.

Otro añadido importante seria desarrollar una aplicación móvil más completa y robusta de forma que amplíe la experiencia del usuario final. No solamente podría ser una aplicación que muestra incidencias, sino, una aplicación capaz de mostrar datos de interés muy variados, y que fuera el usuario en función de sus preferencias, eligiera la información mostrada por la aplicación.

Tras finalizar la aplicación nos hemos dado cuenta que podríamos optimizarla para muchos más usuarios. En primer lugar contamos con la autoescalabilidad de Bluemix, pero podríamos ahorrar gastos si optimizamos la aplicación de la siguiente manera. Tal y como funciona la aplicación actualmente, el sistema solamente se inicializa si la aplicación móvil realiza una petición al servicio. Una posible mejora seria que el servicio esté constantemente buscando incidencias y publicándolas en un broker MQTT. Los usuarios interesados se suscribirían a estos brokers.

Para comprobar la potencia de las PaaS en cuanto a escalabilidad, se podría realizar una prueba con muchos dispositivos para verificar esta funcionalidad. El inconveniente es que estamos ante una PaaS y para realizar esta prueba necesitaríamos pagar por el uso de la aplicación, ya que las características iniciales de la versión gratuita no requieren la autoescalabilidad.

Glosario de acrónimos

- **IOT**: Internet of Things
- **IaaS**: Infraestructure as a Service
- **PaaS**: Platform as a Service
- **SaaS**: Software as a Service
- **GPS**: Global Positioning System
- **MQTT**: Message Queue Telemetry Transport
- **S-Range**: Short Range
- **L-Range**: Long Range
- **RFID**: Radio Frequency Identification
- **M2M**: Machine to Machine
- **WSN**: Red de Sensores Inalámbricos (Wireless Sensor Network)
- **TIC**: tecnologías de la información y comunicación
- **IP**: Internet Protocol
- **TI**: Tecnología de la información
- **TIC**: Tecnología de la comunicación e información
- **URL**: Uniform Resource Locator
- **HTTP**: Hypertext Transfer Protocol
- **REST**: Representational State Transfer
- **JSON**: JavaScript Object Notation

Bibliografía

[1] Honbo Zhou. *The Internet of Things in the Cloud: a Middleware Perspective*. CRC Press, 1 edition, 2012.

[2] Annie Ferrari, David Blanco, and Elena Valdecasa. Cloud computing. retos y oportunidades. *Estudio de investigación*, pages 12–81, 2012.

[3] Katherin Cruz Valencia. Historia del cloud computing. *REVISTA DE INFORMACIÓN TECONOLOGÍA Y SOCIEDAD*, 2015.

[4] ProEspacio (Asociación Española de Empresas del Sector Espacial). Explotación de datos científicos. calculando por las nubes.

[5] Peter Mell and Timothy Grance. The nist definition of cloud computing. *Recommendations of the National Institute of Standards and Technology*, 2011.

[6] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the clouds: A berkeley view of cloud computing. *Technical Report No. UCB/EECS-2009-28*, pages 4–5, 2009.

[7] Luis Joyanes Aguilar. *Computación en la nube: Estrategias de cloud computing en las empresas*. Alfaomega Grupo Editor S.A., 2012.

[8] Marta Beltrán Pardo and Fernando Sevillano Jaén. *Cloud Computing: tecnología y negocio*. Ediciones Nobel, S.A., 2013.

[9] Fran Berman, Geoffrey Fox, and Anthony J.G. Hey. *Grid Computing: Making the Global Infrastructure a Reality*. Wiley, 2003.

[10] Birman K. *Reliable Distribuited Systems: Technologies, Web Services and Applications*. Springer-Verlag, 2005.

[11] FUNCOAS y Plataforma Tecnologica Green TIC Ametic. Situacion y retos de las green tic en espana.

[12] Sosinsky B. *Cloud Computing Bible*. Wiley Pubblishing, Inc, 2011.

[13] Rittinghouse J. W. and Ransome J. F. *Cloud Computing. Implementation, Managment and Security*. CRC Press, 2011.

[14] Lombardi Flavio and Di Pietro Roberto. *Secturity for Cloud Computing*. Boston : Artech House, 2015.

[15] 2011 Company web site. SoftLayer Technologies, Inc. Retrieved August 28. About us.

[16] Duncan C. E. Winn. *Getting Started with Cloud Foundry*. O'Reilly Media, 1 edition, 2016.

[17] Matt Adamson. 2016 comparison matrix report - platform as a service (paas). *Cloud Platforms - Solutions Review*, 2016.

A

Experiencia con IBM Bluemix

Mi experiencia con IBM Bluemix ha sido muy gratificante. Gracias a Bluemix, he podido aprender y comprender el funcionamiento de plataformas como servicio (PaaS) utilizando casos reales de uso mediante los servicios ofrecidos. Los primeros pasos han sido los más complicados ya que era la primera vez que me encontraba con este tipo de entorno de desarrollos. He participado en muchos talleres y meetups organizados por IBM para conocer el funcionamiento de de esta PaaS.

La documentación en inglés es muy clara y además viene con muchos ejemplos para entender el funcionamiento de los servicios. Como hemos estado utilizando la plataforma desde sus inicios, nos hemos encontrado durante este tiempo con servicios que utilizábamos, pero que por algunos motivos, estos servicios se han ido eliminando. Aunque pueda parecer un aspecto negativo, esto nos ha permitido aprender a reaccionar rápidamente ante los cambios en la plataforma. Cuando un servicio se va a eliminar o se va a añadir un nuevo servicio, se notifica a todos los usuarios sobre esa circunstancia.

Mi experiencia con Bluemix va a continuar debido a que es muy fácil elaborar tus propios proyectos, ya que te proporcionan todo lo necesario para desplegar un servicio en producción y nosotros como desarrolladores solamente debemos preocuparnos de la codificación.